DigitalPersona, Inc.

U.are.U SDK

Version 2

Developer Guide

digitalPersona.

Technical Support

To access DigitalPersona technical support resources, go to http://www.digitalpersona.com/support.

Phone support is available at (877) 378-2740 (US) or +1 650-474-4000 (outside the US).

Feedback

Although the information in this guide has been thoroughly reviewed and tested, we welcome your feedback on any errors, omissions, or suggestions for future improvements. Please contact us at

TechPubs@digitalpersona.com

or

DigitalPersona, Inc.
720 Bay Road, Suite 100
Redwood City, California 94063
USA
(650) 474-4000
(650) 298-8313 Fax

Document Publication Date: November 08, 2012 (version 2.2 with JavaPOS and OPOS)

Table of Contents

Introduction 1

The U.are.U SDK allows you to add fingerprint recognition to your application. Significant features of the U.are.U SDK include:

- Performing 1-to-many fingerprint identification automatically
- Support for ANSI and ISO fingerprint images and minutiae data
- Redesigned API for easier programming
- New FingerJet Engine that has met the PIV performance thresholds for fingerprint minutiae data generation required by NIST, with a faster Matcher.

The SDK provides support for developing with Windows (Visual Studio) and Linux with the following languages/frameworks: (X indicates a supported combination)

	Target Platform Supported		
API	**Linux**	**Windows**	**Windows CE**
C/C++	**X**	**X**	**X**
Java	**X**	**X**	
.NET		**X**	**X**
ActiveX		**X**	
JavaPOS	**X**	**X**	
OPOS		**X**	

DigitalPersona supports (and this documentation assumes) development on Windows for target devices based on Windows or Windows CE and development on Linux for target devices based on Linux.

What's New in 2.2

New features in version 2.2 include:

- **Support for U.are.U 5100 fingerprint reader**
- **New JavaPOS API** -- a fully JavaPOS-compliant API, an upgrade from the U.are.U UPOS for JavaPOS SDK. The new JavaPOS API implements JavaPOS, but is based on the U.are.U framework and the internal architecture of the U.are.U SDK. Since the JavaPOS API implements the JavaPOS spec, the terminology and processes of that API are somewhat different than the other APIs in the U.are.U SDK. If you are upgrading an existing application, see *Upgrading from U.are.U UPOS for OPOS/JavaPOS* on *page 26* for more details. The U.are.U JavaPOS API is described in a new chapter of this developer guide: *JavaPOS* on *page 75*.

- **New OPOS API** -- a fully OPOS-compliant API, an upgrade from the U.are.U UPOS for OPOS SDK. The new OPOS API is based on the U.are.U framework and the internal architecture of the U.are.U SDK. Since the OPOS API implements the OPOS spec, the terminology and processes of that API are somewhat different than the other APIs in the U.are.U SDK. If you are upgrading an existing application, see *Upgrading from U.are.U UPOS for OPOS/JavaPOS* on *page 26* for more details. The U.are.U OPOS API is described in a new chapter of this developer guide: *OPOS* on *page 84*.

How the U.are.U Manuals are Organized

The U.are.U documentation set consists of a family of books:

- The U.are.U SDK Developer's Guide (this manual) describes the architecture and organization of the SDK plus an overview of language/framework support.
- Platform Guides for Linux, Windows and Windows CE provide details about installation, requirements and code samples for devices based on Linux, Windows and Windows CE operating systems, respectively.

In addition, we provide Javadoc and Doxygen documentation that provides the detail of method parameters and data structures for the .NET, ActiveX and Java APIs.

Getting Updated Documentation

If you are viewing this developer guide from the download package for the U.are.U SDK, you may want to check online at our website at

http://www.digitalpersona.com/Support/Reference-Material/DigitalPersona-SDK-Reference-Material/

for the latest version of this document.

Target Audience

This manual is aimed at developers who already have a working knowledge of their development environment and their chosen reader platform.

For the C/C++ API, we assume you have a working knowledge of the C++ language.

For .NET developers, we assume that you know how to develop for .NET and understand .NET concepts like static classes, `IDisposable` and `IEnumerable`.

For ActiveX, we assume that you know how to develop with ActiveX.

For Java, we assume a working knowledge of the Java language.

For OPOS and JavaPOS, we assume knowledge of the UPOS (OPOS and JavaPOS) specification version 1.13 and familiarity with Point of Sale applications and developing with OPOS/JavaPOS.

Chapter Overview

Chapter 1, Introduction (this chapter), provides an overview of the features and standards compliance of the U.are.U SDK.

Chapter 2, Background, contains a brief introduction to fingerprint recognition and fingerprint biometrics.

Chapter 3, Developing Applications describes the context and issues for developing applications using the U.are.U SDK. This chapter shows how data flows among the U.are.U SDK components, and includes typical workflows.

Chapter 4, Working with Fingerprint Readers contains information about working with fingerprint reader hardware.

Chapter 5, Converting from Previous SDKs discusses how to upgrade your application to use the current API functions and data formats, as well as a mapping between the One Touch terminology and the terminology used in the U.are.U SDK.

Chapter 6, Using the SDK provides an overview of the SDK and describes how to generate API documentation using Doxygen.

Chapter 7, The C/C++ APIs provides an overview of the functions in the C libraries.

Chapter 8, The Java API describes the interfaces for the Java libraries.

Chapter 9, The .NET API describes the classes and methods in the .NET libraries as well as the .NET controls.

Chapter 10, ActiveX describes the interfaces for the ActiveX libraries and controls.

Chapter 11, JavaPOS describes the JavaPOS API including data management, implementation notes, error codes and exceptions.

Chapter 12, OPOS describes the OPOS API including data management, implementation notes, error codes and exceptions.

Chapter 13, Application Notes provides additional suggestions for getting your application to work.

Chapter 14, Glossary lists the terminology specific to fingerprint recognition applications and to the U.are.U SDK.

Standards Compliance

DigitalPersona U.are.U SDK version 2 Fingerprint Image Data (FID) format is compliant with

- ANSI INSITS 381-2004
- ISO/IEC 19794-4:2005

DigitalPersona U.are.U SDK Version 2 Fingerprint Minutiae Data (FMD) format is compliant with

- ANSI INSITS 378-2004
- ISO/IEC 19794-2:2005

DigitalPersona U.are.U SDK version 2 can process FIDs and raw images generated by other systems as long as they are compliant with the aforementioned standards and:

- Are Uncompressed
- Have horizontal and vertical resolutions that are the same (square pixels)
- Have 8 bit per pixel

For multiview image records, we also require:

- No more than 16 views per finger
- All views in a multiview image must have the resolution and same pixel dimensions (Width x Height)
- We do not support unknown finger positions (finger position = 0).

DigitalPersona U.are.U SDK version 2 can process minutiae data generated by other systems as long as they are compliant with the aforementioned standards and meet this criterion:

- Horizontal and Vertical resolutions are the same (square pixels)

Background 2

In this chapter, we discuss the basics of fingerprint recognition. This chapter is not intended to be exhaustive, rather we're going to give you enough background knowledge to develop your own application more effectively.

For a more detailed overview, we recommend **Handbook of Fingerprint Recognition** by D. Maltoni, M. Maio, A. Jain, and S. Prabhakar, published by Springer, 2nd edition, 2009.

Biometrics

Identifying individuals based on their distinctive anatomical (fingerprint, face, iris, hand geometry) and behavioral (signature, voice) characteristics is called ***biometrics***. Because biometric identifiers cannot be shared or misplaced, they intrinsically represent an individual's identity. Biometrics is quickly becoming an essential component of effective identification solutions. Recognition of a person by their body, then linking that body to an externally established "identity", forms a powerful authentication tool.

Biometric identification helps to reduce fraud, and enhance user convenience. Among the different biometric identification methods, fingerprint recognition technology has a good balance of qualities including accuracy, throughput, size and cost of readers, maturity of technology and convenience of use, making it the dominant biometric technology in commercial applications.

Features of Biometric Technology

Biometric solutions offer many advantages that other technologies cannot provide.

- Uniqueness - Fingerprints from each one of our ten fingers is distinctive, different from one another and from those of other persons. Even identical twins have different fingerprints.
- Convenience - Users no longer have to remember multiple, long and complex, frequently changing passwords or carry multiple keys.
- Non-repudication - Ensures the user is present at the point and time of recognition and later cannot deny having accessed the system.
- Non-transferable - Cannot be shared, lost, stolen, copied, distributed or forgotten unlike passwords, PINs, and smart cards.
- Proven - Long history of successful use in identification tasks - the U.S. and other countries have extensive real-world experience with fingerprint recognition. Fingerprints have been used in forensics for well over a century and there is a substantial body of scientific studies and real world data supporting the distinctiveness and permanence of fingerprints.

The Basics of Fingerprint Identification

The skin on the inside surfaces of our hands, fingers, feet, and toes is “ridged” or covered with concentric raised patterns. These ridges are called friction ridges and they provide friction making it easier for us to grasp and hold onto objects and surfaces without slippage. The many differences in the way friction ridges are patterned, broken, and forked make ridged skin areas, including fingerprints, distinctive.

The distinctiveness of fingerprints is well established. The underlying biological persistence of fingerprint characteristics is also a well established fact reported in various fingerprint studies conducted in different scientific fields over the past century.

Fingerprint Characteristics

Fingerprint ridges are not continuous, straight ridges. Instead, they are broken, forked, interrupted or changed directionally. The points at which ridges end, fork, and change are called ***minutiae points*** which provide distinctive, identifying information.

The most common properties of fingerprint minutiae points are:

1. Type

There are several types of minutiae points, some of which are listed below. The most common are ridge endings and ridge bifurcations.

- Ridge ending – the abrupt end of a ridge
- Ridge bifurcation – a single ridge that divides into two ridges
- Short ridge, or independent ridge – a ridge that commences, travels a short distance and then ends
- Island – a single small ridge inside a short ridge or ridge ending that is not connected to all other ridges
- Ridge enclosure – a single ridge that bifurcates and reunites shortly afterward to continue as a single ridge
- Spur – a bifurcation with a short ridge branching off a longer ridge
- Crossover or bridge – a short ridge that runs between two parallel ridges
- Delta – a Y-shaped ridge meeting
- Core – a U-turn in the ridge pattern.

2. Direction

3. Position

Issues in Fingerprint Recognition Technology

In a perfect world, it would be a simple matter to determine whether two fingerprints were from the same finger-- the images would be identical or they would not. However,

- Even though our fingerprints do not change over time, the fingerprint *images* can vary a lot, especially for some people. For example, certain skin conditions and wear due to manual labour can affect fingerprint images. This makes fingerprint recognition a very challenging problem that does not have a perfect solution. As the result, captured fingerprint images are often not a perfect match to the stored image from the same finger.
- Fingerprint images from two different fingers of two different people can look similar, especially when, because of worn fingerprints or temporary creases, there is very little information left about the actual fingerprint. The larger the population your are working with, the more likelihood of similar fingerprint images.

Fingerprint recognition software needs to address these issues. We'll discuss that more in later sections of this guide.

Developing Applications 3

This chapter describes the process of developing fingerprint recognition applications, including:

- An overview of how fingerprint recognition works
- Data structures and data flows among components
- Typical workflows
- Design issues and tradeoffs.

IMPORTANT: The JavaPOS and OPOS APIs conform to the UPOS specification and therefore do not have exactly the same data structures and workflow as the other APIs in the U.are.U SDK. See Chapter 11, *JavaPOS* and Chapter 12, *OPOS* for details.

How Fingerprint Recognition Works

Fingerprint recognition works in two stages:

1. First, users are enrolled with the system--their fingerprints are captured and stored in a database.
2. Next, when a person needs to be given access (e.g., to open a door or to log in to a computer), they simply scan their finger on the fingerprint reader.

In terms of application development, this typically requires the developer to build the following components:

1. An application for people to ***enroll***:
   - Captures multiple fingerprints for at least two fingers from a fingerprint reader.
   - Checks image quality to ensure that a good quality scan is obtained.
   - Extracts the fingerprint minutiae.
   - Saves the fingerprint in a database.
2. A service(s)/application(s) that ***identifies/verifies*** people:
   - Captures a fingerprint from a fingerprint reader.
   - Extracts the fingerprint minutiae.
   - Compares fingerprint with enrolled fingerprints to ***identify*** a user from a list or ***verify*** a specific user.

This SDK provides fingerprint capture, extraction, enrollment and identification/verification functions to help you develop these components.

Understanding the Data Flow

When building a fingerprint recognition application, the data flow consists of:

1. Capture a ***Fingerprint Image*** (scan) from the fingerprint reader. The resulting ***Fingerprint Image Data (FID)*** contains one or more fingerprint images, called a ***Fingerprint Image Views (FIVs)***. A typical FID for fingerprint recognition applications contains only one FIV but we also support multiple views (e.g., if there are multiple fingers from one individual or multiple images from a single finger stored in a single FID).

   Each FIV (fingerprint) is approximately 140K in size.
2. Extract the fingerprint features. During extraction, ***Fingerprint Minutiae Data (FMD)*** is created, with each fingerprint stored in a ***Fingerprint Minutiae View (FMV)*** in the FMD. A FMV in a FMD takes no more than 1.5K (maybe less depending on the fingerprint). FMDs are used for identifying users in a collection and verifying specific users.

The ANSI and ISO standards permit multiple views but the U.are.U SDK creates only single-view FIDs and FMDs.

This data flow is shown in *Figure 1* below.

Note that in previous SDKs, DigitalPersona products used a proprietary format where enrollment fingerprints were stored as *templates* and fingerprint to be identified/verified were created as *feature sets*. ANSI and ISO standard data formats are the same for templates and feature sets, so while the current SDK supports the DigitalPersona proprietary data format for backward compatibility, new applications should be developed using ANSI and ISO standard data formats and the data flow as shown below. (The data flow for legacy data may be different, as described in *Working with Legacy Data and Deprecated Data Formats* on *page 24*.)

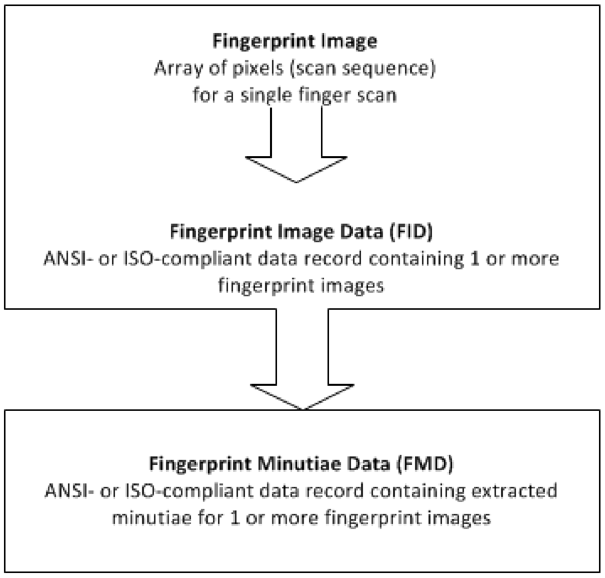

Fingerprint Image
Array of pixels (scan sequence) for a single finger scan
Fingerprint Image Data (FID)
ANSI- or ISO-compliant data record containing 1 or more fingerprint images
Fingerprint Minutiae Data (FMD)
ANSI- or ISO-compliant data record containing extracted minutiae for 1 or more fingerprint images

Figure 1. Data flow used by U.are.U SDK in fingerprint recognition

Workflow - Enrollment Application

During the enrollment process, one or more fingers are scanned for each person. We recommend that you enroll at least two fingers (more is recommended) because in the event of an accident or injury to one finger, another enrolled finger can be used to identify the individual.

The enrollment application needs to perform the following steps to enroll a single finger from a user:

Step One - Initialization

Initialize the library. Discover the available readers and open a connection to the reader.

Step Two - Capture, Extract and Enroll

1. Begin the enrollment process.
2. Capture a series of fingerprint scan(s); for each scan,
   - Create a FID,
   - Extract fingerprint minutiae and create the FMD,
   - Add the FMD to the pool of FMDs for enrollment.
3. Continue to capture fingerprints until the enrollment process has enough FMDs to complete the enrollment. (The enrollment functions evaluate the FMDs and select the best image -- typically several scans are required.)
4. Create the enrollment FMD and release resources.

Step Three - Store Data

Store the enrollment FMD. Many applications keep only the enrollment FMD because of space constraints or policy decisions. You cannot use FIDs for identification, so even if you choose to keep the FIDs, you must also store the FMD for each individual.

Notes on Enrollment

Before storing, you may want to check for existing entries that match the new entry -- applications like law enforcement, banking or voting registration, may not allow duplicate enrollments.

The capture/extract minutiae part of the enrollment process is the same as for capturing/extracting minutiae for the purpose of verifying/identifying users. If you wish, you can enroll users without using the enrollment functions (by simply capturing, extracting minutiae and storing the resulting FMD). However we recommend that you use the enrollment functions to create the best quality enrollment FMDs.

The enrollment process is slightly different in each API. Consult the chapters that describe the various APIs to determine the specifics for your language. For JavaPOS and OPOS, the enrollment process is described in the specification.

Workflow - Identifying/Verifying

Fingerprint recognition involves two types of operation:

- **Identification** - Comparing a fingerprint against the database of enrolled fingerprints and confirming that the fingerprint is enrolled (e.g., to open a door there many be many authorized users).
- **Verification** - Comparing a fingerprint against a specific user's enrolled fingerprint(s) to verify a specific person's identity (e.g., when the user types their name and then uses a fingerprint rather than a password).

To perform these operations, your application needs to do the following steps:

Step One - Initialization

Initialize the library. Discover the available readers and open a connection to a reader.

Step Two - Capture and Extract

1. Wait for a fingerprint. When a fingerprint is detected, capture the image and create an FID.
2. Extract fingerprint minutiae and create an FMD.

This sequence is exactly the same as for the capture/extraction process during enrollment.

Step Three - Identify/Verify

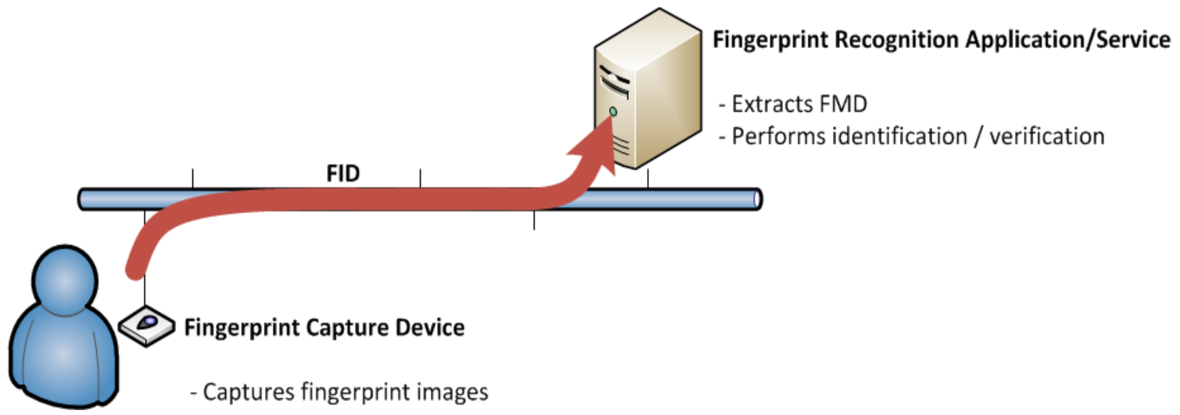
Call the appropriate function to ***verify*** a specific person OR to ***identify*** a valid user.

Design Issues for your Application

Distributed Processing and Data Flow

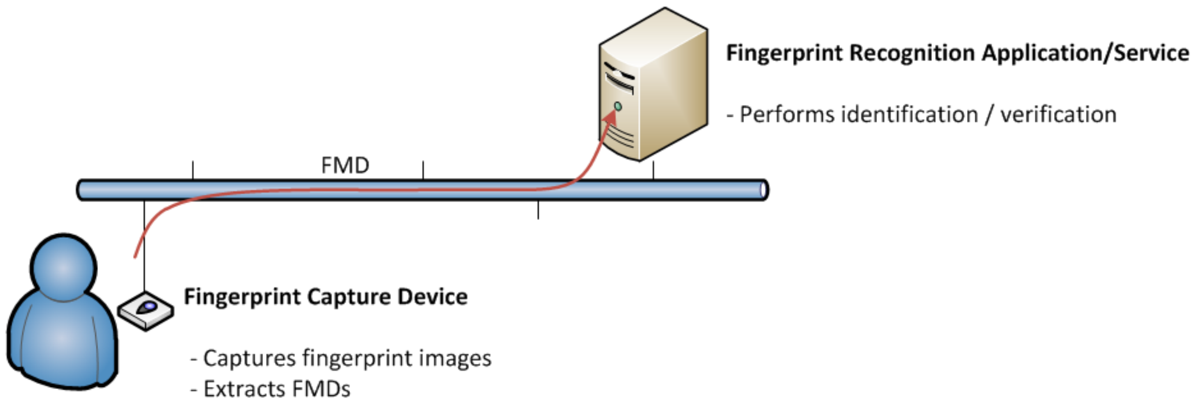
Depending on the capabilities of your fingerprint reader, you can capture FIDs and send them to another machine for processing OR the fingerprint reader can extract the FMD and transmit only the much smaller FMD files. Thus the application can be designed in these two ways:

1. The fingerprint capture device can simply capture a fingerprint image and transmit the image to a server for processing as shown in the image below. Since FIDs are large (around 100K - 140K), this means that you need a faster connection, but there is less computing power required by the fingerprint reader.


Fingerprint Recognition Application/Service
- Extracts FMD
- Performs identification / verification
FID
Fingerprint Capture Device
- Captures fingerprint images

Optionally, the fingerprint capture device can compress the image before transmitting it.

2. Another alternative is to develop software for the fingerprint capture device to capture the fingerprint image AND extract the fingerprint features to create a FMD. The FMD is then transmitted to the server for processing, as shown below. FMDs are 1.5K or less and so require less bandwidth and speed.


Fingerprint Recognition Application/Service
- Performs identification / verification
FMD
Fingerprint Capture Device
- Captures fingerprint images
- Extracts FMDs

Data Compression

If you use compression to save bandwith or storage space, there are two options:

1. U.are.U provides compression using Wavelet Scalar Quantization (WSQ). WSQ is a wavelet-based compression standard developed by NIST specifically for fingerprint data.
2. You can compress the data using a lossless compression such as JPEG2000.

WSQ Compression

WSQ compression allows compression of 15:1 to 12:1. For more details, see *Wavelet Scalar Quantization (WSQ) Compression* on *page 34*.

JPEG2000 Compression

If you are using JPEG2000 as a compression algorithm, choose 12 as the quality setting. This allows lossless compression at around 10:1. The U.are.U SDK does not support JPEG2000 compression directly, so you would have to capture the fingerprint as a raw image and compress it to JPEG2000. After transmission, you must convert the JPEG2000 file back into a raw image for minutiae extraction and identification/verification.

NIST Fingerprint Image Quality (NFIQ)

The NFIQ score indicates the quality of a fingerprint sample. The U.are.U SDK provides NFIQ calculation using code developed by NIST. The NFIQ score is in the range 1 – 5, with 1 being the best and 5 being not suitable for feature extraction. For more details, see *NIST Fingerprint Image Quality (NFIQ)* on *page 35*.

Determining an Acceptable Level of Error

When identifying fingerprints, you want to identify strictly enough that you do not let unauthorized people have access (false positives) but also do not inconvenience legitimate users by rejecting their fingerprints (false negatives). Note that some people will always experience more false rejections -- the rate of false negatives is a statistical measure, but *individuals* may experience higher rejection rates, based on their specific fingerprint characteristics.

There is a trade-off between the frequencies of false positive and false negative errors. Applications have control over this trade-off by specifying the threshold for the required degree of similarity between two fingerprint images in order to call it a match. **When choosing the identification threshold, note that increasing the false positive error rate by a factor of 100 will reduce the false negative error rate only approximately by a factor of 2.**

There will always be some false negatives. As the result, every practical fingerprint recognition system should have an alternative means to establish and prove identity, without using fingerprints.

Setting the Error Threshold when Identifying a Fingerprint in a Collection

When a fingerprint is scanned, the first step is to identify the fingerprint against a set of stored FMDs.

- If you are trying to confirm that a user is allowed access, you will want to identify the fingerprint against all the valid FMDs that you have stored.
- If you are trying to confirm the identity of a specific person, you must identify against all FMDs for that individual (typically at least two fingers are stored for each user).

The identification function compares a FMV against a collection of FMDs to produce the candidate list. You can specify the maximum desired number of candidates: a smaller number can make the execution faster. The most similar candidates are returned closer to the beginning of the candidate list.

Your ***threshold*** determines the trade-off between false positive and false negative error rates where:

- 0 = no false positives
- `maxint` (#7FFFFFFF or 2147483647) = fingerprints do not match at all
- Values close to 0 allow very few false positives; values closer to `maxint` allow very poor matches (a lot of false positives) in the candidate list. The table below shows the ralationship between the threshold values and the false positive identification error rates observed in our test. Note: the actual false positive identification error rates in your deployment may vary.

Your Threshold	Corresponding False Positive Identification Rate	Expected number of False Positive Identifications	Numeric Value of Threshold
.001 * maxint	.1%	1 in 1,000	2147483
.0001 * maxint	.01%	1 in 10,000	214748
.00001 * maxint	.001%	1 in 100,000	21474
1.0e-6 * maxint	.0001%	1 in 1,000,000	2147

For many applications, a good starting point for testing is a threshold of 1 in 100,000. If you want to be conservative, then you will want to set the threshold lower than the desired error rate (e.g., if you want an error rate that does not exceed 1 in 100,000, you might set the threshold to 1 in 1,000,000).

Defining the Data Retention Policy

After a fingerprint scan, a FID is created, which contains the actual image. Each fingerprint image takes roughly 140K of storage. With modern computers, that is not a huge amount, but each enrolled user may have several fingerprints scanned. Multiplied by the number of potential users, this can add up to a fair amount of data.

To identify fingerprints, you must extract the fingerprint characteristics to create a FMD, which is < 1.5K per fingerprint.

Some applications choose to retain the full image records, but other companies discard the image record and retain only the FMD. Note that if you discard the image record, you cannot reconstruct the original fingerprint image from the FMD, the FMD is only useful for identifying/verifying fingerprints. From time to time

DigitalPersona may release a new version of the U.are.U SDK that will provide improved accuracy in the feature extraction process. If you do not retain the fingerprint images, you will not be able to redo the feature extraction using the new version of the SDK/runtime in order to take advantage of these improvements.

Specifying the Fingers to Be Scanned

When you enroll people into your system, you will usually want more than one finger enrolled. This allows for injury and makes it easier for people who have fingerprints that are difficult to recognize. For some applications you may want to scan all ten fingers or take multiple fingerprint impressions for individual fingers.

A typical policy would be to require both index fingers or both thumbs to be scanned. Thumbs are typically the worst in terms of recognition accuracy and thumbs require different capture devices with larger area and different ergonomics. Where possible, avoid any industrial design forcing users to use thumbs.

The preferred approach is to enroll, at a minimum, both index and both middle fingers. Middle fingers are usually the best, probably because people have almost as good dexterity with middle fingers as with index fingers, and yet middle fingers have fingerprints that are typically less worn than index fingers.

Some solutions are ergonomically designed in a way that only the right or only the left hand can be used conveniently. In this case the right hand is better (probably because the majority of people are right handed), and at least three fingers need to be enrolled: index, middle and ring fingers.

Ergonomics and the correct finger placement are extremely important. Poor ergonomics can easily increase the false negative identification rate by a factor or 5 to 10. The system users need to be aware of correct finger placement for best results.

Optimizing Fingerprint Applications

To identify a fingerprint against a large database can take a considerable amount of time and create unacceptable delays between the fingerprint scan and the user authorization. Identifying fingerprints will be faster if the database of fingerprints is in memory rather than retrieved from disk. If you have a large database of users, you may need to provision your server with an appropriate amount of RAM to handle the searches.

This SDK is not optimized for large scale identification. If you are developing such an application, you may want to contact DigitalPersona to get help selecting the technology that will best fit your needs.

To ensure the fastest response time, you must weigh whether it will be faster to transmit the image record to a server for FMD extraction or whether it is faster to extract the FMD on the reader and transmit only the FMD to the server for identification/verification.

For readers that are used by a limited number of people (e.g., kiosks or pharmacy cabinets), you may have the device identify fingerprints against a limited set of FMDs. However this requires that you keep the FMDs in the device in sync with your central database, to ensure that new employees are able to gain access and departing employees' privileges are revoked quickly.

Working with Fingerprint Readers 4

The U.are.U SDK works with the following fingerprint readers:

- U.are.U 4000B Fingerprint Reader Rev. 100 (all platforms) and Rev. 101 48Mhz (Windows CE only)
- U.are.U 4500 Fingerprint Reader Rev. 103
- U.are.U 5100 Fingerprint Reader

The 4xxx series readers have limited support for streaming. Not all platforms have complete support for all readers -- check the platform guides for more details.

Fingerprint readers can lose their calibration as a result of changes in temperature, humidity and ambient light. Humidity is the most important environmental factor affecting calibration. U.are.U fingerprint readers are self-calibrating, but you still might want to set up your application to check periodically that no additional calibration is needed.

If the fingerprint reader is not giving clear readings:

- Try cleaning it:
  - For the 4xxx readers, clean the gel surface with sticky tape. Gently dab it with the sticky side of the tape. Do not rub it with paper and do not get it wet.
  - For the 5xxx readers, in addition to sticky tape, you can use a damp wipe. Do not clean with compressed air and do not use industrial cleaners or solvents.
- Make sure that you are touching the fingerprint reader with the pad of your finger, not the tip. The most detail (minutiae) occur roughly midway between the first joint and the tip.
- If your fingers are very dry, try touching your forehead with the pad of the finger you are trying to scan and then rescanning your fingerprint.

Good ergonomics in the mounting of your fingerprint reader can significantly improve the quality of fingerprint scans. Be sure that the reader is mounted in a way that is convenient for users to touch properly, such that the large area of the pad of the finger is captured. The angle at which the reader is mounted as well as the rim around the sensing area can make it difficult for people with large fingers or long fingernails to have proper contact between the pad of the finger and the sensing area.

If your fingerprint reader becomes non-responsive from an electrostatic shock you may need to perform a hardware reset.

Your application should check the reader status between fingerprint scans to ensure that the hardware has not experienced an error condition.

Converting from Previous SDKs 5

Overview of Support for Previous SDKs

The U.are.U SDK (C++, Java, .NET and ActiveX) can exchange data with applications developed using the following DigitalPersona products:

- DigitalPersona Gold SDK/Fingerprint Recognition Software
- DigitalPersona One Touch SDKs

The U.are.U SDK does not support the custom encryption keys that are supported by other DigitalPersona products.

The U.are.U SDK can use data developed using:

- DigitalPersona U.are.U UPOS for JavaPOS
- DigitalPersona U.are.U UPOS for OPOS

Applications developed with U.are.U UPOS for JavaPOS or UPOS for OPOS can be upgraded to use the JavaPOS and OPOS APIs of the U.are.U SDK, since both the previous and the current product support the UPOS specification.

Converting Applications from One Touch SDK

To convert an application from One Touch SDK, install the U.are.U SDK as described above and use this documentation to modify your applications to use the new API.

Be sure to install U.are.U SDK in a new folder on your development machine so that your existing files do not get overwritten.

Note that when you install the U.are.U SDK on the target reader, **the One Touch drivers will be overwritten with new drivers that are compatible with both your existing applications and new applications based on U.are.U SDK**.

The files installed on the target reader include both drivers and SDK files. If you retain the One Touch SDK files on the device, you will need up to an additional 120K for the new U.are.U SDK files. If you retain the old SDK files on the device , you can run applications based on either the old or the new SDK. The two SDKs can coexist and your existing applications can run using the older run-time environment while you update your programs or develop new applications based on the new SDK. Applications based on the old and new SDKs can run on the same hardware, but not simultaneously.

Note that the ***data*** from existing applications based on One Touch SDK is fully compatible with applications based on U.are.U SDK. For more information, see *Working with Legacy Data and Deprecated Data Formats* on *page 24*.

Working with Legacy Data and Deprecated Data Formats

The DigitalPersona Gold SDK and One Touch SDK products used a different format for minutiae data (which were called feature sets). This data format has been deprecated. In this older format, there were three different data types for minutiae data which reflected the intended use of the data:

- ***Pre-registration features*** (Gold SDK), or ***Pre-registration Feature Set*** to be used for Enrollment (OneTouch SDK); Pre-registration features were intended only for creation of Registration features. An application needed to collect four fingerprints of the finger to enroll, extract pre-registration features, and pass it to the SDK to produce registration features.
- ***Registration features*** (Gold SDK), or ***Fingerprint Template*** (OneTouch SDK); Registration features were intended to be stored in a database as an enrolled finger.
- ***Verification features*** (Gold SDK), or ***Feature Set*** (One Touch SDK). Verification features are what is compared to the Registration features when a user swipes a finger.

In the previous SDKs, you could not compare two Feature Sets, you could only compare a Feature Set against an Fingerprint Template. The U.are.U SDK removes that distinction and when you create data with ANSI/ISO data formats, all minutiae data is stored in an FMD, whether produced by the feature extraction functions or the enrollment functions.

You may also choose to use the legacy DigitalPersona format for your data. For new applications, we recommend a standardized (ANSI or ISO) format.

When passing an array of fingerprint templates into the identification function, all the templates in the array should be in the same format. If you have existing data in the legacy format, you must continue to use the legacy format for all of your data.

The identification and comparison functions can work with the legacy data formats listed above.

The U.are.U SDK supports the legacy data formats to allow for continued support of old applications, but we encourage you to convert your data to a newer format for new applications.

Changes in U.are.U Terminology from Legacy Usage (Gold and One Touch)

This section describes changes in terminology from the One Touch documentation and other DigitalPersona products.

Old Term	New Term	Explanation
Fingerprint authentication	Fingerprint verification	Change in industry standard terminology.
Fingerprint registration	Fingerprint enrollment	Change in industry standard terminology.
Fingerprint sensor (referring to a fingerprint capture device)	Fingerprint reader	Erroneous usage: Sensors are a *component* of certain types of *fingerprint capture devices*, they are not themselves *fingerprint capture devices*.
Match Matching Matching score	Compare Comparison Comparison score	Change in industry standard terminology.
Performance	Recognition accuracy	More accurate terminology.
Fingerprint feature set Fingerprint template	Fingerprint minutiae data	*Fingerprint templates* (*fingerprint features* stored for enrolled fingers) and *fingerprint feature sets* (images used for verification and identification) have been replaced with *fingerprint minutiae data* in the U.are.U family of the SDKs only. With standards-based data formats in the U.are.U SDKs, all *fingerprints* are stored in the same format.
ROC curve	DET curve	More accurate terminology. We have never used ROC curves, however our DET curves were sometimes erroneously called ROC curves in the past.
FAR	FMR or FNMR as appropriate	The definition of *FAR* is application-specific. In some applications *FAR* is similar to *FMR* while in other applications, *FAR* is similar to *FNMR*.
FRR	FMR or FNMR as appropriate	The definition of *FRR* is application-specific. In some applications *FRR* is similar to *FNMR*, while in the others *FRR* is similar to *FMR*.

Upgrading from U.are.U UPOS for OPOS/JavaPOS

This new version of the OPOS and JavaPOS APIs are backward-compatible. However the following new features may or may not affect your existing applications:

- **The default data format has a longer header.** The default data format is now fully compliant with UnifiedPOS 1.13. The default data format is the same as the old format except that the template header has additional bytes of information, i.e., a 45-byte header, instead of the previous 10-byte header for JavaPOS and 12-byte header for OPOS. If you do not update your application to specify a specific format, new templates will be created with 45-byte headers. Enrollment, identification and verification will continue to work with both new templates with 45-byte headers and previous templates that have the shorter headers. See *Working with Fingerprint Data in JavaPOS* on *page 75* and *Working with Fingerprint Data in OPOS* on *page 84* for more information.
- **New FAR Security Setting.** We have tightened the FAR security setting to align with our current corporate standard. The new standard is 100 times more strict. Please update your code accordingly. We recommend that you allow the FAR setting to be configurable at run-time within your application. See Chapter 3, *Determining an Acceptable Level of Error,* on *page 19* for more information on setting FAR.
- **Enrollment no longer stops at four retries.** The previous enrollment process would fail after four unsuccessful scans. If your application is dependent on having exactly four tries, then you may need to adjust your code.
- **Errors and exceptions may be handled differently.** Because of the extensive architectural changes to this version of the OPOS and JavaPOS APIs, exceptions and error codes may not be identical. You should double check carefully that the new API handles errors and exceptions as expected for your application.

For specific details about upgrading your existing application, consult the platform guide for your target hardware.

Using the SDK 6

What's In the SDK?

The SDK consists of:

1. C/C++ API -- C libraries that conform to ANSI.C99 (http://en.wikipedia.org/wiki/C99):
   - **DP Capture API** - for capturing fingerprints
   - **FingerJet Engine API** - for extracting fingerprint characteristics and identifying/verifying fingerprints
2. .NET API -- .NET class libraries
   - **DP .NET API** - for capturing and comparing fingerprints
   - **DP .NET Controls** - simple interface for enrollment and identification, based on OneTouch interface
   - **DP ActiveX Library** - for capturing and comparing fingerprints using the ActiveX wrapper with .NET
   - **DP ActiveX Controls** - simple interface for enrollment and identification, based on OneTouch interface
3. ActiveX API and controls -- ActiveX class libraries
   - `DPXUru.dll` – ActiveX API library
   - `DPCtlXUru.dll` – ActiveX GUI controls
4. Java API -- Java class libraries
   - `dpuaru.jar` - library classes and interfaces for working with readers and the FingerJet Engine
5. JavaPOS API -- class libraries that implement a JavaPOS-compliant API (per the JavaPOS 1.13 specification), as a wrapper to the U.are.U Java API:
   - **`dpjavapos.jar`** - library classes and interfaces for working with readers
   - JavaPOS Device Service object, which can be used with any JavaPOS Device Control for the Biometrics device category
6. OPOS API -- classes that implement an OPOS-compliant API (per the UPOS 1.13 specification), as a wrapper to the U.are.U C/C++ API:
   - `dpServiceObject.dll` – a custom implementation of the OPOS data service
   - **OPOSBiometrics** – a custom implementation of the OPOS biometrics control
7. Run-time components:
   - Capture driver and SDK layer
   - FingerJet Engine run-time

FingerJet Engine

FingerJet Engine is a module that extracts fingerprint characteristics from image records to create FMDs and compares FMDs to confirm identity. FingerJet Engine has met the PIV performance thresholds for fingerprint minutiae data generation required by NIST.

We support a maximum of 16 views in a single FID or FMD during authentication. All views in a single record must have the same resolution. We do not support unknown finger positions (finger position = 0).

The C/C++ APIs 7

The C/C++ APIs are available for Linux, Windows and Windows CE. This chapter provides an overview of the APIs. For details of using the API on a specific reader platform, consult the appropriate Platform Guide.

Detailed documentation for the API is contained in the header files. You can simply read the header files or you can view the Doxygen files. Consult the platform guide for details of where the Doxygen files are located.

The API is thread-safe.

DP Capture API

The DP Capture API consists of library management, reader management, capturing and streaming.

Library Management

Table 1. Library management functions

Function	Description
dpfpfdd_init	Initialize the library (allocate system resources and initialize data). **This must be the first function called.**
dpfpdd_exit	Release the library and its resources.
dpfpdd_version	Query the library version. **This is the only function that can be called before dpfpdd_init or after dpfpdd_exit.** This returns the DP Capture API library version (not the U.are.U SDK version). This is analogous to the dpfj_version function which returns the version of the FingerJet library file.

Fingerprint Capture Device Management

Table 2. Hardware management functions

Function	**Description**
dpfpdd_query_devices	Discover connected devices.
dpfpdd_open	Open a device. This function establishes an exclusive link to the device; no other processes will be able to use the device until you close it. The application *must* open the device before use.
dpfpdd_close	Close a device.
dpfpdd_get_device_status	Get the status for a device. You would normally check the device status between captures to ensure that the device is functioning and there are no error conditions.
dpfpdd_get_device_capabilities	Query a device for information on capabilities.
dpfpdd_set_parameter	Change a device or driver parameter
dpfpdd_get_parameter	Query a device or driver parameter
dpfpdd_calibrate	Calibrate a reader. Some readers are self-calibrating. Ambient light or temperature can affect calibration, for some readers. Calibration can take several seconds.
dpfpdd_reset	Do a hardware reset on the reader. Hardware resets are typically needed only after a hardware problem (e.g., the device is unplugged or receives an electrostatic shock). Hardware resets typically only take a few milliseconds.

Capturing Fingerprints

The `dpfpdd_capture` function captures a fingerprint image for

- Enrollment (as part of the process described on page *32)*
- Identifying users with `dpfj_identify`
- Verifying a specific user identity with `dpfj_compare`

Table 3. Fingerprint capture functions

Function	Description
dpfpdd_capture	Capture a fingerprint image from the reader. This function signals the reader that a fingerprint is expected, and waits til a fingerprint is received.
dpfpdd_cancel	Cancel a pending capture

Streaming Fingerprints

Not all readers support streaming mode. To determine if a specific reader supports this feature, check the value of `can_stream_image` in `DPFPDD_DEV_CAPS`, as returned by `dpfpdd_get_device_capabilities()`.

The streaming methods are:

Table 4. Streaming Functions

Function	Description
dpfpdd_start_stream	Start streaming mode
dpfpdd_get_stream_image	Capture a fingerprint image from the streaming data
dpfpdd_stop_stream	End streaming mode

FingerJet Engine API

The FingerJet API contains functions that extract features from FIDs to create FMDs, identify/verify FMDs and convert FMDs to different formats.

Library Management

Table 5. Library version verification function

Function	Description
dpfj_version	Query the library version. This returns the FingerJet library version (not the U.are.U SDK version). This is analogous to the dpfpfdd_version function which returns the version of the DP Capture API library file.

Extract FMD

Table 6. Feature extraction functions

Function	Description
dpfj_create_fmd_from_raw	Creates FMD from raw image
dpfj_create_fmd_from_fid	Creates FMD from ANSI, ISO or DigitalPersona legacy format FID

Identify Fingerprint

This function is described in detail on page *17* and page *19*.

Table 7. Fingerprint identify function

Function	Description
dpfj_identify	Identify a FMD: given an array of FMDs, this function returns an array of candidates that match the original fingerprint (a FMV within a FMD) within the threshold of error. Supported formats are: Gold SDK, One Touch SDK, ANSI and ISO.

Enrollment

The enrollment functions allow you to enroll a finger to create a FMD that you can store in your database. For ANSI/ISO formats, the enrollment functions create FMDs. For legacy DigitalPersona format, the enrollment functions create a fingerprint template.

The typical process would be:

1. Call `dpfj_start_enrollment.`
2. Capture a fingerprint scan and extract an FMD, using the standard functions (`dpfpdd_capture` to capture and `dpfj_create_fmd_from_fid` or `dpfj_create_fmd_from_raw` to extract).
3. Call `dpfj_add_to_enrollment` to add the fingerprint to the potential pool.
4. Repeat the previous two steps until `dpfj_add_to_enrollment` returns a flag indicating the pool of FMDs is now sufficient to create an enrollment FMD.
5. Create the enrollment FMD with `dpfj_create_enrollment_fmd` and release resources by calling `dpfj_finish_enrollment`.
6. Store the enrollment FMD in your database. Some applications like voting, banking and law enforcement require that you check for duplicate fingerprints before storing a new fingerprint in the database.

Table 8. Enrollment Functions

Function	Description
dpfj_start_enrollment	Begin the enrollment process and allocate resources.
dpfj_add_to_enrollment	Add the FMD to the pool of FMDs for enrollment and return a flag indicating that the enrollment is ready (enough FMDs have been received to create the enrollment FMD)
dpfj_create_enrollment_fmd	Create FMD for enrolled finger
dpfj_finish_enrollment	Release resources used during enrollment process

Format Conversion

Table 9. Conversion functions to convert FMDs from legacy formats or between supported formats

Function	Description
dpfj_fmd_convert	Convert FMDs from ANSI to ISO format and vice versa.
dpfj_dp_fid_convert	Convert legacy DigitalPersona image (Gold SDK and One Touch SDK) to ANSI or ISO images

Advanced Diagnostics

The majority of applications should use the `dpfj_identify` function to implement both identification and verification. However, in a few special cases, e.g., using multi-modal biometrics, or doing statistical risk assessment, the `dpfj_compare` function allows you to compare two FMVs to determine their actual degree of dissimilarity. This is useful for accuracy testing and diagnostics and is not intended to be used in final applications for actual fingerprint recognition. The `dpfj_compare` function returns a ***dissimilarity score*** with values:

- 0 = fingerprints are NOT dissimilar (i.e., they MATCH perfectly).
- `maxint` (#7FFFFFFF or 2147483647) = fingerprints are completely dissimilar (i.e., DO NOT match).
- Values close to 0 indicate very close matches, values closer to `maxint` indicate very poor matches.

The table below shows the relationship between the scores returned from `dpfj_compare` and the false match error rates observed in our test. The dissimilarity score distribution is estimated based on our internal testing, and may not be representative of the actual rate that will be observed in deployment.

Dissimilarity Score	**False Match Rate**
2147483	.1%
214748	.01%
21474	.001%
2147	.0001%

Table 10. Fingerprint verification

Function	**Description**
dpfj_compare	Compare two FMDs; supported formats are: Gold SDK, One Touch SDK, ANSI and ISO.

Wavelet Scalar Quantization (WSQ) Compression

The U.are.U SDK provides compression of fingerprint images with the WSQ algorithm. This compression is available in two ways:

1. Using code developed by NIST. This code is included within the U.are.U C/C++ API.
2. Using the WSQ1000 SDK from Aware, Inc. on the target system. If the WSQ 1000 SDK is installed, the U.are.U functions will call the WSQ 1000 functions to do compression and decompression instead of the native NIST-based code.

   DigitalPersona does not redistribute the Aware WSQ1000 SDK, it must be acquired and installed separately.

At publication time, the WSQ compression functions are available in the U.are.U SDK for Windows and U.are.U SDK for Linux on x86 and x64 platforms. WSQ compression is not available in the U.are.U SDK for Windows CE and in the U.are.U SDK for Linux on the ARM platform.

The typical sequence to compress an image would be:

- `dpfj_start_compression`
- `dpfj_set_wsq_bitrate` or `dpfj_set_wsq_size` - Set the desired size for the compressed image
- `dpfj_compress_fid` or `dpfj_compress_raw` - Compress the FID or raw image
- `dpfj_get_processed_data` - Retrieve the compressed image
- `dpfj_finish_compression`

The typical process to decompress an image would be:

- `dpfj_start_compression`
- `dpfj_expand_fid` or `dpfj_expand_raw` - Decompress a FID or raw image
- `dpfj_get_processed_data` - Retrieve the expanded image
- `dpfj_finish_compression`

.

Table 11.WSQ compression and decompression functions

Function	**Description**
dpfj_start_compression	Initiates WSQ compression and allocates resources.
dpfj_finish_compression	Releases resources.
dpfj_set_wsq_bitrate dpfj_set_wsq_size	These two specify the same thing: the size of the resulting compressed image, and can be used interchangeably. Setting the bitrate at 0.75bpp to 0.9 bpp allows compression of 15:1 to 12:1. The parameter `tolerance_aw` sets the tolerance as required by the Aware WSQ1000 SDK and it is ignored when using NIST algorithm.
dpfj_compress_fid dpfj_compress_raw	Compress an FID or raw image, according to the requested size.
dpfj_get_processed_data	Retrieve the image that was just compressed/expanded.
dpfj_expand_fid dpfj_expand_raw	Expand a previously compressed FID or raw image.

NIST Fingerprint Image Quality (NFIQ)

The U.are.U SDK calculates NFIQ scores for fingerprint images. This calculation is available in two ways:

1. Using code developed by NIST. This code is included within the U.are.U C/C++ API.

2. Using the WSQ1000 SDK from Aware, Inc. on the target system. If the WSQ 1000 SDK is installed, the U.are.U functions will call the WSQ 1000 functions to do NFIQ calculation instead of the native NIST-based code.

   DigitalPersona does not redistribute the Aware WSQ1000 SDK, it must be acquired and installed separately.

The NFIQ calculation functions are available in U.are.U SDK for Windows, U.are.U SDK for Linux on x86 and x64 platforms. NFIQ calculation functions are not available in the U.are.U SDK for WindowsCE and in the U.are.U SDK for Linux on the ARM platform.

Table 12.NFIQ Functions

Function	**Description**
dpfj_quality_nfiq_from_fid dpfj_quality_nfiq_from_raw	Calculate the NFIQ score of a FID or raw image. NFIQ scores range from 1 to 5 with 1 being the best quality and 5 indicating that the image is not suitable for feature extraction.

The Java API 8

The Java API is built as a wrapper to the C/C++ API. The Java API is available for Linux and Windows. This chapter provides an overview of the API. For details of using the API on a specific reader platform, consult the appropriate Platform Guide.

The Java API is considerably simpler to use than the C/C++ APIs and therefore generally results in:

- Easier data management
- Easier enrollment
- Faster development

Importing the U.are.U Java package

The U.are.U Java library classes and interfaces are aggregated into `dpuaru.jar`. To use the U.are.U Java library functionality import the `com.digitalpersona.uareu.*` package, and make sure to include `dpuareu.jar` into your classpath.

Getting Detailed Documentation

This chapter provides an overview of the main methods in the Java API. For a complete description of method parameters, the Javadoc documentation for the Java libraries is provided. Consult the platform guide for details of where the Javadoc files are located.

Using the Package

Main Access Point

The main access point to the U.are.U Java library is the `UareUGlobal` class. This is a static class, which allows you to acquire references to the classes related to fingerprint readers and to the FingerJet Engine:

- **To acquire a reference to `ReaderCollection`** use the `GetReaderCollection()` method. To destroy `ReaderCollection`, (release all system resources associated with readers and make readers available for other processes) use the `DestroyReaderCollection()` method.
- **To acquire a reference(s) to individual readers**, use the `ReaderCollection` object, which is a collection of objects of type `Reader`.
- **To acquire a reference to the FingerJet Engine** use the `GetEngine()` method. The engine does not use or allocate any system resources except memory and does not have to be destroyed explicitly.

UareUException

The `UareUException` interface describes exceptions specific to the U.are.U SDK.

Getting a List of Available Readers

The `ReaderCollection` interface provides a list of the readers connected to the machine. A list of available readers can be acquired any time with `GetReaders()` method.

Working with Readers

Each attached reader is represented with a `Reader` object. The `Reader` interface allows:

- Querying reader description and capabilities,
- Acquiring status of the reader,
- Capturing fingerprints,
- Starting and stopping video stream, and
- Resetting and calibrating the reader.

The main methods are:

Table 13. Hardware management methods

Function	**Description**
GetDescription	Get the description of a reader. The description is available at any time (even if the device is not open). This is the only method that can be called before the `open()` method. Returns an object of type `Description` holding information about the reader hardware.
Open	Open a device and return the device capabilities. This method establishes an exclusive link to the device; no other processes will be able to use the device until you close it. The application *must* open the device before use.
GetStatus	Get the status for a device. You would normally check the device status between captures to ensure that the device is functioning and there are no error conditions. Returns an object of type `ReaderStatus` which describes the current status of the reader.

Table 13. Hardware management methods

Function	**Description**
GetCapabilities	Get the capabilities of a device. Returns an object of type `Capabilities` which describes what the reader can do.
Calibrate	Calibrate a device. Some devices are self-calibrating. Ambient light or temperature can affect calibration, for some devices. Calibration can take several seconds.
Reset	Do a hardware reset on the reader. Hardware resets are typically needed only after a hardware problem (e.g., the device is unplugged or receives an electrostatic shock). Hardware resets typically only take a few milliseconds.
Close	Close a reader and release the resources associated with the reader.

Capturing Fingerprints

The `Capture` function captures a fingerprint image for

- Enrollment (as part of the process described on page *16)*
- Identifying users with `Identify`
- Verifying a specific user identity with `Compare`

The primary fingerprint capture methods are:

Table 14. Fingerprint capture functions

Function	**Description**
Capture	Captures a fingerprint image from the open reader. This function signals the reader that a fingerprint is expected, and blocks until a fingerprint is received, capture fails or the reader times out.
CancelCapture	Cancel a pending capture

Streaming Fingerprints

Not all readers support streaming mode. To determine if a specific reader supports this feature, get the reader capabilities with the `GetCapabilities` method and check the value of the field `can_stream`.

The streaming methods are:

Table 15. Streaming Functions

Function	Description
StartStreaming	Put the reader into streaming mode. In this mode, the application must call `GetStreamImage()` to acquire images from the stream.
GetStreamImage	Capture a fingerprint image from the streaming data. After this function returns, the reader remains in streaming mode. Frame selection, scoring and other image processing are not performed by this function.
StopStreaming	End streaming mode

Accessing the FingerJet Engine

The `Engine` interface provides functionality to

- Extract fingerprint features (create FMDs),
- Identify and compare FMDs, and
- Create enrollment FMDs.

Refer to *Understanding the Data Flow* on *page 15* for a description of enrollment and comparison terminology and data flow.

Creating FMDs from images

The `CreateFmd()` method can be used in two ways:

1. Extract fingerprint minutiae from a raw image and create a FMD.
2. Extract fingerprint minutiae from a FID and create a FMD.

The following limitations are applied to the raw images and FIDs:

- 8 bits per pixel
- no padding
- square pixels (horizontal and vertical dpi are the same)

The size of the resulting FMD will vary depending on the minutiae in a specific fingerprint.

Identification and Comparison

`Identify()` identifies a single FMD against an array of FMDs. This function takes as inputs:

- A single view in an FMD
- An array of FMDs (each FMD can contain up to 16 views) to compare
- The desired number of candidates to return
- The threshold for False Positive Identification Rate (FPIR) that is permitted

Each time a view has a score lower than the threshold FPIR, that view is marked as a possible candidate. Then when all possible candidates are identified (i.e., they meet the threshold), they are ranked by their score. Finally, the function returns as many candidates as requested, based on the candidates with the lowest dissimilarity score. For a discussion of setting the threshold as well as the statistical validity of the dissimilarity score and error rates, consult *NIST Fingerprint Image Quality (NFIQ)* on *page 19*.

`Compare()` takes two single views from two FMDs and returns a dissimilarity score indicating the quality of the match.

The majority of applications should use the `Identify` method to implement both identification and verification. However, in a few special cases, e.g., using multi-modal biometrics, or doing statistical risk assessment, the `Compare` method allows you to compare two FMVs to determine their actual degree of dissimilarity. This is useful for accuracy testing and diagnostics and is not intended to be used in final applications for actual fingerprint recognition. The `Compare` method returns a ***dissimilarity score*** with values:

- 0 = fingerprints are NOT dissimilar (i.e., they MATCH perfectly).
- `maxint` (#7FFFFFFF or 2147483647) = fingerprints are completely dissimilar (i.e., DO NOT match).
- Values close to 0 indicate very close matches, values closer to `maxint` indicate very poor matches.

The table below shows the relationship between the scores returned from `Compare` and the false match error rates observed in our test. The dissimilarity score distribution is estimated based on our internal testing, and may not be representative of the actual rate that will be observed in deployment.

Dissimilarity Score	**False Match Rate**
2147483	.1%
214748	.01%
21474	.001%
2147	.0001%

Table 16. Methods for Verifying and Identifying Fingerprints

Function	**Description**
Compare	Compare two FMDs; supported formats are: Gold SDK, One Touch SDK, ANSI and ISO.
Identify	Identify a FMD: given an array of FMDs, this function returns an array of candidates that match the original fingerprint (a FMV within a FMD) within the threshold of error. Supported formats are: Gold SDK, One Touch SDK, ANSI and ISO.

Enrollment

`CreateEnrollmentFmd()` creates and returns an enrollment FMD. It takes as input a reference to an object of type `EnrollmentCallback`. The client must implement `EnrollmentCallback.GetFmd()`. This method acquires and returns an FMD to add to the enrollment. The engine calls `EnrollmentCallback.GetFmd()` as many times as needed in order to create an enrollment FMD.

Normally, the client application will implement `EnrollmentCallback.GetFmd()` to provide the onscreen UI to capture fingerprints from the reader, extract features using `CreateFmd()`, and return an enrollment FMD. If the user wants to cancel the enrollment, `EnrollmentCallback.GetFmd()` should return `null`.

The .NET API 9

The .NET API is built as a wrapper to the C/C++ APIs. The .NET API is available for Windows and Windows CE. This chapter provides an overview of the API. For details of using the API on a specific reader platform, consult the appropriate Platform Guide.

The .NET API is considerably simpler to use than the C/C++ APIs and therefore generally results in:

- Easier data management
- Easier enrollment
- Faster development

The .NET API also implements additional features:

- Serialization
- Pre-built GUI controls for enrollment and identification to quickly get you started

Importing the U.are.U .NET package

The U.are.U .NET library classes are aggregated into the following class libraries:

- **DP .NET API** - for capturing and comparing fingerprints
- **DP .NET Controls** - simple interface for enrollment and identification, based on OneTouch interface

Getting Detailed Documentation

This chapter provides an overview of the main methods in the .NET API. For a complete description of method parameters, the Doxygen documentation for the .NET libraries is provided. Consult the platform guide for details of where the Doxygen files are located.

Using the Package

Main Access Points

- **To acquire a reference to `ReaderCollection`** use the `GetReaders()` method. To destroy `ReaderCollection`, (release all system resources associated with readers and make readers available for other processes) use the `Dispose()` method.
- **To acquire a reference(s) to individual readers**, use the the `ReaderCollection` object, which is a collection of objects of type `Reader`.

- **To work with the FingerJet Engine,** instantiate one of the classes: `Comparison`, `Importer`, `FeatureExtraction` or `Enrollment`. For example, to start enrolling fingerprints using the FingerJet engine, call the static method, `Enrollment.CreateEnrollmentFmd().` This method takes as input an enumeration that specifies the format the enrollment template should be in, e.g., ANSI or ISO. This method also takes an `IEnumerable<Fmd>` object. See *IEnumerables in the .NET Wrapper* on *page 45* for more information on `IEnumerable<Fmd>.`

SDKException

The `SDKException` class describes exceptions specific to the U.are.U SDK.

Serialization

The .NET API provides the ability to convert a FMD or FID into a format that can be read serially. Serialization allows you to easily transmit and store data as byte strings, streams or XML. This allows you to transmit the data or save it to standard file systems. The XML format can be used within HTML for building browser-based applications.

To return the data to its original format, *deserialization* is also provided.

Note that serialized FMDs and FIDs include the version number of the .NET wrapper that was used for the serialization. If you attempt to deserialize with an older version of the .NET wrapper, the results may not be correct, so when deserializing, the .NET wrapper will throw an SDKException if the serialization was done using a later version of the wrapper.

Serialization in the .NET Wrapper occurs using the default `System.Xml.Serialization.XmlSerializer`. The name of the root element of the generated XML is the same as the object type. Each public member of the the object is represented as an XML element. Raw byte data is encoded as a Base-64 string by `XmlSerializer`.

Deserialization uses the same `XmlSerializer`. The XML that was serialized is fed into `XmlSerializer` where the elements with values become object members with values of the same name as the element.

IEnumerables in the .NET Wrapper

The .NET wrapper uses the IEnumerable<T> interface (as specified here: http://msdn.microsoft.com/en-us/library/9eekhta0.aspx). An IEnumerable<T> is a data structure of type T that can be iterated through. An array is one example of IEnumerable<T>, because it can be iterated, or looped through.

An advantage of IEnumerable is that many different data structures can be used instead of requiring a very specific type of data structure, such as a 1-dimensional array of type T.

Perhaps the most important advantage of IEnumerables is that whatever is responsible for sending an IEnumerable to a method may use special symantics such as '`yield return`' to return only a single item at a time. This helps performance, and also helps programmers simplify their code by allowing .NET wrapper methods to pull only what is necessary for the operation.

For example, `Enrollment.CreateEnrollmentFmd(IEnumerable<Fmd>)` can iterate through a user's FMDs until enough data is captured and then create an enrollment FMD, as opposed to having more data than is necessary. Each iteration can can cause a capture workflow to occur which causes '`yield return`' to return a single item to Enrollment.

NOTE: The `yield return` feature of IEnumerable is available in C# but **NOT** in VB.NET.

For more information on IEnumerable<T>, consult the Microsoft documentation pages listed below.

Topic	**Microsoft Documentation**
IEnumerable<T>	http://msdn.microsoft.com/en-us/library/9eekhta0.aspx
Yield	http://msdn.microsoft.com/en-us/library/9k7k7cf0(v=vs.80).aspx

Working with Readers

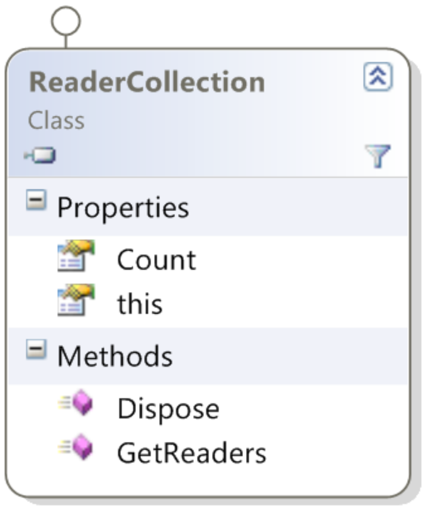
Readers are accessed through the `ReaderCollection` object, which is of type `IEnumerable<Reader>`.

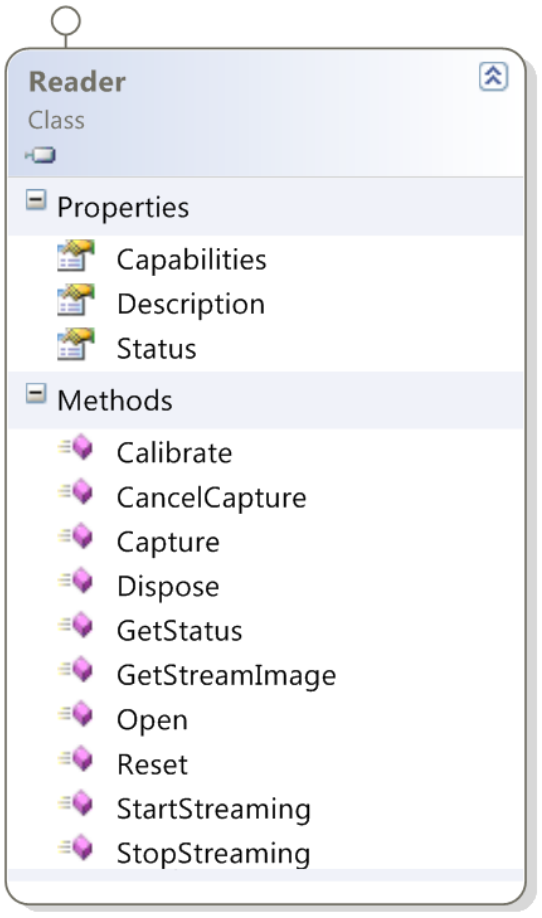
Each attached reader is represented with a `Reader` object. The `Reader` interface allows:

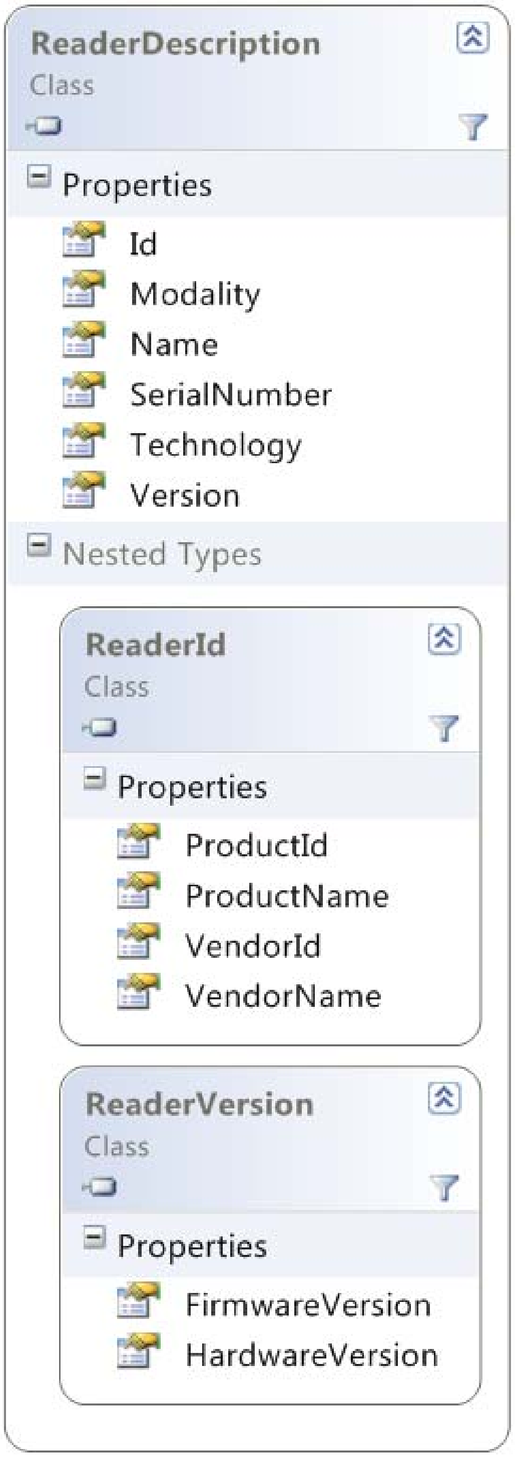
- Querying reader description and capabilities,
- Acquiring status of the reader,
- Capturing fingerprints,
- Starting and stopping image stream, and
- Resetting and calibrating the reader.

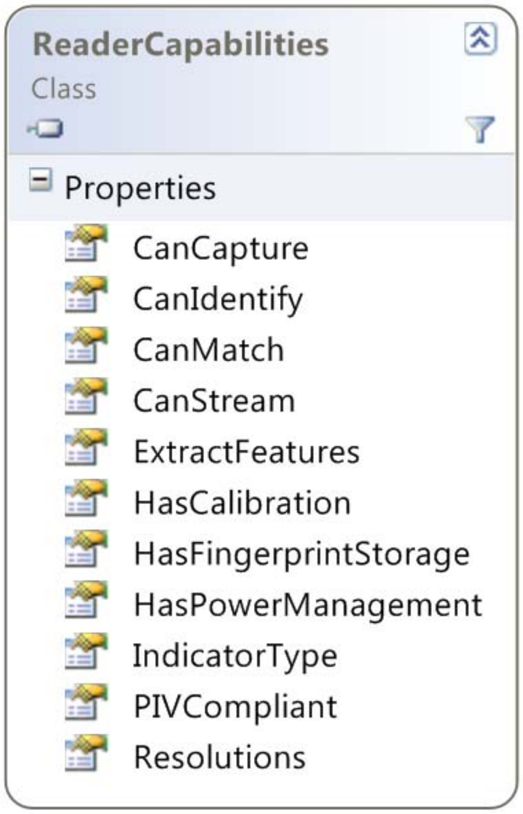
To acquire a reader, call `ReaderCollection.GetReaders()` which returns the attached readers. Iterate through this list to look at the description object for the desired reader. Once finished with all of the `Reader` objects in the `ReaderCollection`, ensure that you call the `Dispose()` method to un-instantiate the `ReaderCollection` object and release system resources.

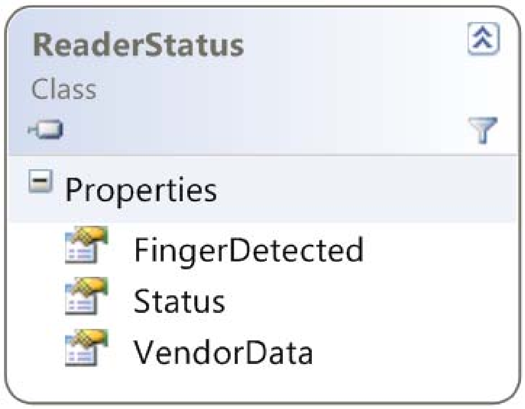
The UML diagrams below show the `ReaderCollection` and `Reader` classes.


ReaderCollection
Class
Properties
Count
this
Methods
Dispose
GetReaders


Reader
Class
Properties
Capabilities
Description
Status
Methods
Calibrate
CancelCapture
Capture
Dispose
GetStatus
GetStreamImage
Open
Reset
StartStreaming
StopStreaming


ReaderDescription
Class
Properties
Id
Modality
Name
SerialNumber
Technology
Version
Nested Types
ReaderId
Class
Properties
ProductId
ProductName
VendorId
VendorName
ReaderVersion
Class
Properties
FirmwareVersion
HardwareVersion


ReaderCapabilities
Class
Properties
CanCapture
CanIdentify
CanMatch
CanStream
ExtractFeatures
HasCalibration
HasFingerprintStorage
HasPowerManagement
IndicatorType
PIVCompliant
Resolutions


ReaderStatus
Class
Properties
FingerDetected
Status
VendorData

The main methods for managing reader hardware in the `Reader` class are:

Table 17. Hardware management methods

Function	**Description**
Open	Open a device and return the device capabilities. This method establishes an exclusive link to the device; no other processes will be able to use the device until you close it. The application *must* open the device before use.
GetStatus	Get the status for a device. You would normally check the device status between captures to ensure that the device is functioning and there are no error conditions. Returns an object of type `ReaderStatus` which describes the current status of the reader.
Calibrate	Calibrate a device. Some devices are self-calibrating. Ambient light or temperature can affect calibration, for some devices. Calibration can take several seconds.
Reset	Do a hardware reset on the reader. Hardware resets are typically needed only after a hardware problem (e.g., the device is unplugged or receives an electrostatic shock). Hardware resets typically only take a few milliseconds.
Dispose	Close device, release memory, remove child objects.

Capturing Fingerprints

The `Capture` function captures a fingerprint image for

- Enrollment (as part of the process described on page *16)*
- Identifying users with `Identify`
- Verifying a specific user identity with `Compare`

The primary fingerprint capture methods are:

Table 18. Fingerprint capture functions

Function	**Description**
Capture	Captures a fingerprint image from the open reader. This function signals the reader that a fingerprint is expected, and blocks until a fingerprint is received, capture fails or the reader times out.
CancelCapture	Cancel a pending capture

Streaming Fingerprints

Not all readers support streaming mode. To determine if a specific reader supports this feature, check the value of the reader property `CanStream`.

The streaming methods are:

Table 19. Streaming Functions

Function	**Description**
StartStreaming	Put the reader into streaming mode. In this mode, the application must call `GetStreamImage()` to acquire images from the stream.
GetStreamImage	Capture a fingerprint image from the streaming data. After this function returns, the reader remains in streaming mode. Frame selection, scoring and other image processing are not performed by this function.
StopStreaming	End streaming mode

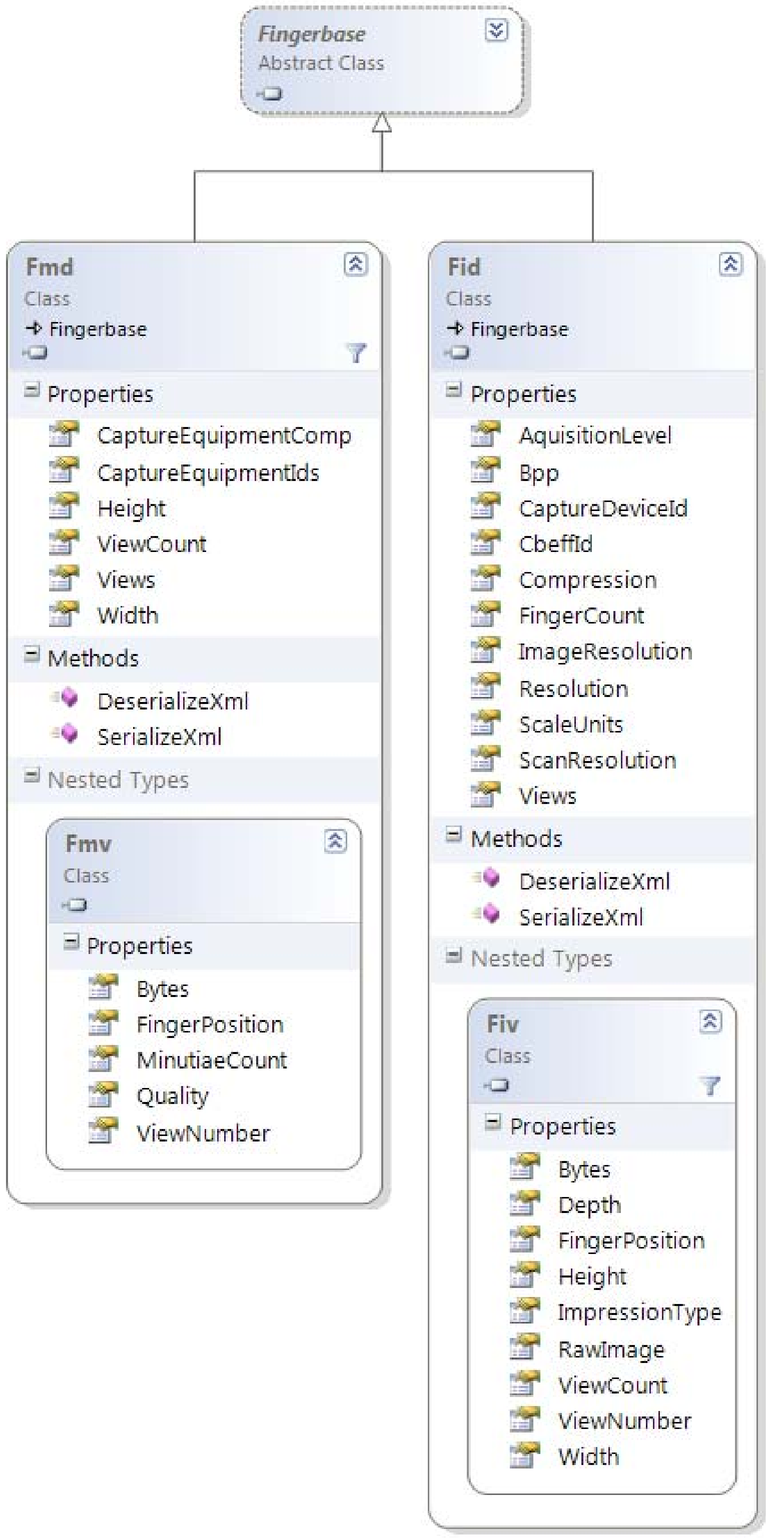
Managing Fingerprint Data

The .NET API implements fingerprint data as shown in the UML diagram below and on the next page.

In addition to the usual methods for working with data, this class includes the methods below for serializing and deserializing (as described in *Importing the U.are.U .NET package* on *page 44*.

Table 20. Serializing and Deserializing

Function	Description
SerializeXml	Static method which serializes FMDs and FIDs into an XML string. Note that raw data is encoded as base-64.
DeserializeXml	Static method which de-serializes XML-encoded FMDs and FIDs back into a U.are.U FMD or FID data structure.



Fingerbase
Abstract Class
Fmd
Class
Fingerbase
Properties
CaptureEquipmentComp
CaptureEquipmentIds
Height
ViewCount
Views
Width
Methods
DeserializeXml
SerializeXml
Nested Types
Fmv
Class
Properties
Bytes
FingerPosition
MinutiaeCount
Quality
ViewNumber
Fid
Class
Fingerbase
Properties
AquisitionLevel
Bpp
CaptureDeviceId
CbeffId
Compression
FingerCount
ImageResolution
Resolution
ScaleUnits
ScanResolution
Views
Methods
DeserializeXml
SerializeXml
Nested Types
Fiv
Class
Properties
Bytes
Depth
FingerPosition
Height
ImpressionType
RawImage
ViewCount
ViewNumber
Width

Analyzing and Managing Fingerprints (FingerJet Engine)

The FingerJet Engine interfaces provide functionality to

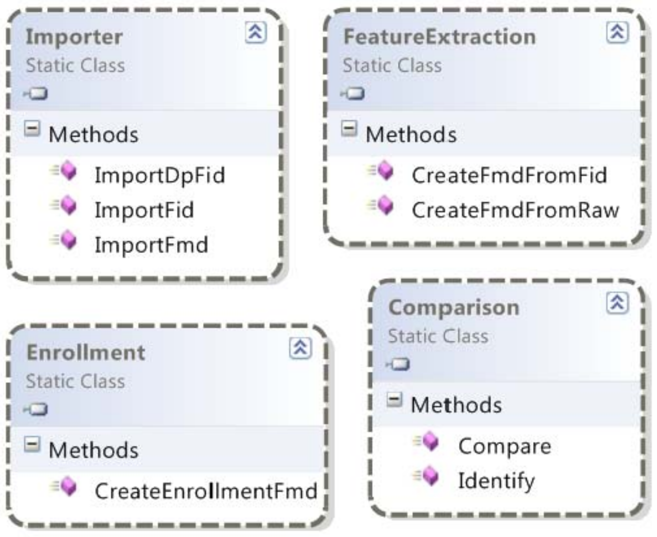
- Extract fingerprint features (create FMDs),
- Identify and compare FMDs, and
- Create enrollment FMDs.

Refer to *Understanding the Data Flow* on *page 15* for a description of enrollment and comparison terminology and data flow.

The Classes for working with data (including identify and verify) are:

Class	**Description**
Comparison	This class allows you to perform: - Verification - Identification
Enrollment	To enroll a user, you must call: CreateEnrollmentFMD Enrollment is implemented as a static class -- you can provide an `IEnumerable<Fmd>` or a function that returns such.
Importer	The U.are.U SDK supports multiple formats for FMDs and FIDs. You can import any supported format.
FeatureExtraction	Extract features from a FID or raw image.

The UML diagram below provides more detail.



Creating FMDs from images

The `CreateFmdFromFid()` method can be used in two ways:

1. Extract fingerprint minutiae from a raw image and create a FMD.
2. Extract fingerprint minutiae from a FID and create a FMD.

The following limitations are applied to the raw images and FIDs:

- 8 bits per pixel
- no padding
- square pixels (horizontal and vertical dpi are the same)

The size of the resulting FMD will vary depending on the minutiae in a specific fingerprint.

Comparing Fingerprints

`Identify()` identifies a single FMD against an array of FMDs. This function takes as inputs:

- A single view in an FMD
- An array of FMDs (each FMD can contain up to 16 views) to compare
- The desired number of candidates to return
- The threshold for False Positive Identification Rate (FPIR) that is permitted

and returns matches as an array of integer pairs which provide the finger index and view index of each match.

Each time a view has a score lower than the threshold FPIR, that view is marked as a possible candidate. Then when all possible candidates are identified (i.e., they meet the threshold), they are ranked by their score. Finally, the function returns as many candidates as requested, based on the candidates with the lowest dissimilarity score. For a discussion of setting the threshold as well as the statistical validity of the dissimilarity score and error rates, consult *NIST Fingerprint Image Quality (NFIQ)* on *page 19*.

`Compare()` takes two single views from two FMDs and returns a dissimilarity score indicating the quality of the match.

The majority of applications should use the `Identify` method to implement both identification and verification. However, in a few special cases, e.g., using multi-modal biometrics, or doing statistical risk assessment, the `Compare` method allows you to compare two FMVs to determine their actual degree of dissimilarity. This is useful for accuracy testing and diagnostics and is not intended to be used in final applications for actual fingerprint recognition. The `Compare` method returns a ***dissimilarity score*** with values:

- 0 = fingerprints are NOT dissimilar (i.e., they MATCH perfectly).
- `maxint` (#7FFFFFFF or 2147483647) = fingerprints are completely dissimilar (i.e., DO NOT match).
- Values close to 0 indicate very close matches, values closer to `maxint` indicate very poor matches.

The table below shows the relationship between the scores returned from `Compare` and the false match error rates observed in our test. The dissimilarity score distribution is estimated based on our internal testing, and may not be representative of the actual rate that will be observed in deployment.

Dissimilarity Score	**False Match Rate**
2147483	.1%
214748	.01%
21474	.001%
2147	.0001%

Table 21. Methods for Verifying and Identifying Fingerprints

Function	Description
Compare	Compare two FMDs; supported formats are: Gold SDK, One Touch SDK, ANSI and ISO.
Identify	Identify a FMD: given an array of FMDs, this function returns an array of candidates that match the original fingerprint (a FMV within a FMD) within the threshold of error. Supported formats are: Gold SDK, One Touch SDK, ANSI and ISO.

Enrollment

`CreateEnrollmentFmd()` creates and returns an enrollment FMD. The method takes as input an enumeration which defines which format to use, e.g., ANSI or ISO. This method also takes as input an `IEnumerable`<Fmd> object type. See *IEnumerables in the .NET Wrapper* on *page 45* for a brief description. As `IEnumerable`<Fmd> pertains to Enrollment, you may either send an array of FMDs to `CreateEnrollmentFmd()` because simple arrays satisfy the `IEnumerable`<Fmd> interface. You may also send a method which returns `IEnumerable`<Fmd>. Here is an example that uses an `IEnumerable`<Fmd> method. To keep the example simple, error checking is removed:

```
// Create an enrollment Fmd.
DataResult<Fmd> enrollmentResult = Enrollment.CreateEnrollmentFmd(
Constants.Formats.Fmd.ANSI,
CaptureExtractFmd()
);
```

Here is a function which captures and extracts an FMD, then returns `IEnumerable`<Fmd>: (the full version of this code is in the code sample)

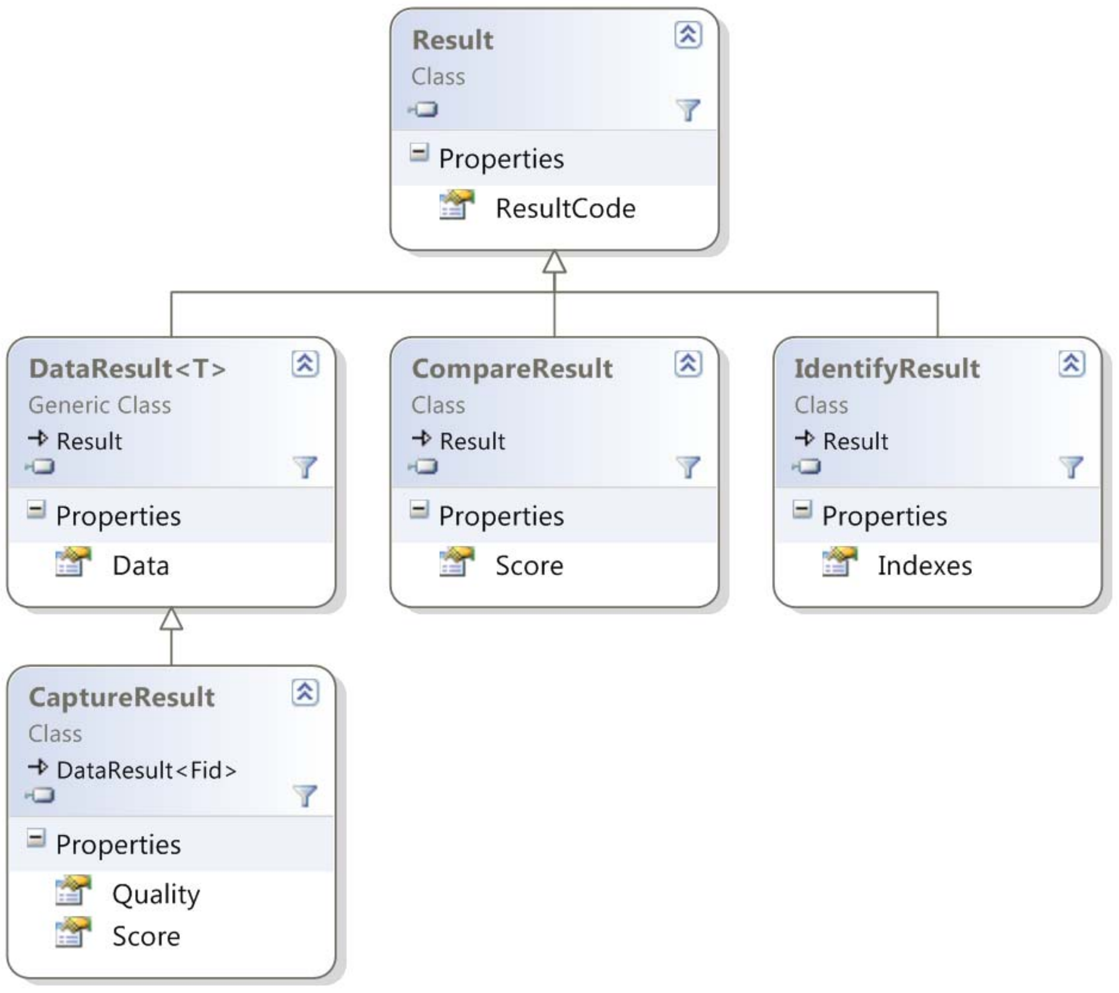
```
// Capture and extract an FMD and return as IEnumerable<Fmd>.
private IEnumerable<Fmd> CaptureAndExtractFmd()
{
  while (true)
  {
      // !!! Get Status and ensure that status is DP_STATUS_READY before continuing
      // Capture a fingerprint.
      CaptureResult captureResult = m_reader.Capture(
          Constants.Formats.Fid.ANSI,
          Constants.CaptureProcessing.DP_IMG_PROC_DEFAULT,
              /* Default processing used. */
          -1,
              /* No timeout. */
          reader.Capabilities.Resolutions[0]
              /* A resolution that is available to reader. */
          );

          // !!! Check for errors, use ‘yield return null; or break;’ to stop.
          yield return convertResult.Data;
  }
}
```

In VB.NET, the `IEnumerable`<T> class does not exist. The recommended method to enroll FMDs using VB.NET is to send `CreateEnrollmentFmd()` four captured FMDs in an array. If four FMDs is not enough, add a fifth captured and converted FMD to the array and send to `CreateEnrollmentFMD()`, and so on until a FMD is created.

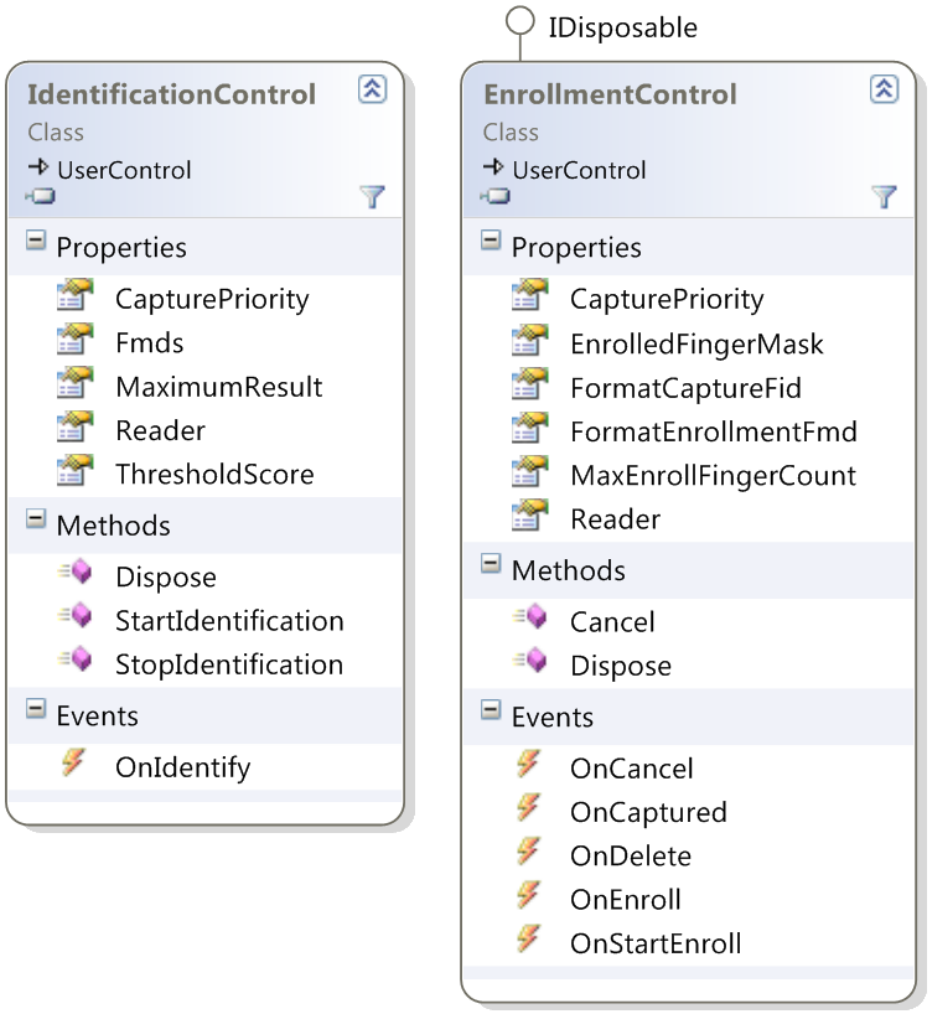
Common Data Structures for Results

The following classes are used when data is returned as a result of a U.are.U .NET operation.



Pre-Built Controls for Enrollment and Identification

In addition to the basic API, the SDK contains two pre-built controls that allow you to quickly add enrollment or identification into your application. These controls provide the same functionality as the DigitalPersona One Touch products.



ActiveX 10

The ActiveX API and controls are built as a wrapper to the C/C++ APIs.

The ActiveX API is available for Windows.

This chapter provides an overview of the API. For details of using the API on Windows-based readers, consult the U.are.U SDK Platform Guide for Windows.

Note that ActiveX is a Microsoft technology and is not supported on Mozilla Firefox and Google Chrome.

Importing the U.are.U ActiveX package

The U.are.U ActiveX library classes are aggregated into two DLLs:

- **`DPXUru.dll`** – ActiveX API library
- **`DPCtlXUru.dll`** – ActiveX GUI controls

Getting Detailed Documentation

This chapter provides an overview of the main methods in the ActiveX API. For a complete description of method parameters, the Doxygen documentation for the ActiveX libraries is provided. Consult the platform guide for details of where the Doxygen files are located.

Using the Package

Main Access Points

The main access point to the U.are.U ActiveX library is the `XReaderCollection` class.

- **To acquire a reference to `XReaderCollection`** use the `GetReaders()` method. To destroy `XReaderCollection`, (release all system resources associated with readers and make readers available for other processes) use the `Dispose()` method.
- **To acquire a reference(s) to individual readers**, use the the `XReaderCollection` object, which is a collection of objects of type `XReader`.
- **To work with the FingerJet Engine,** instantiate one of the classes: `XComparison`, `XImporter`, `XFeatureExtraction` or `XEnrollment`. For example, to start enrolling fingerprints using the FingerJet engine, call the static method, `Enrollment.CreateEnrollmentFmd().` This method takes as input an enumeration that specifies the format the enrollment template should be in, e.g., ANSI

or ISO. This method also takes an `IEnumerable<Fmd>` object. See *IEnumerables in the .NET Wrapper* on *page 45* for more information on `IEnumerable<Fmd>.`

SDKException

The `SDKException` class describes exceptions specific to the U.are.U SDK.

Serialization

The ActiveX API provides the ability to convert a FMD or FID into a format that can be read serially. Serialization allows you to easily transmit and store data as byte strings, streams or XML. This allows you to transmit the data or save it to standard file systems. The XML format can be used within HTML for building browser-based applications.

To return the data to its original format, *deserialization* is also provided.

Note that serialized FMDs and FIDs include the version number of the ActiveX API that was used for the serialization. If you attempt to deserialize with an older version of the API, the results may not be correct, so when deserializing, the API will throw an SDKException if the serialization was done using a later version of the wrapper.

Serialization in the API occurs using the default `System.Xml.Serialization.XmlSerializer`. The name of the root element of the generated XML is the same as the object type. Each public member of the the object is represented as an XML element. Raw byte data is encoded as a Base-64 string by `XmlSerializer`.

Deserialization also uses `XmlSerializer`. The XML that was serialized is fed into `XmlSerializer` where the elements with values become object members with values of the same name as the element.

Working with Readers

Readers are accessed through the `XReaderCollection` object, which is a collection of `XReader` objects.

Each attached reader is represented with a `XReader` object. The `XReader` interface allows:

- Querying reader description and capabilities,
- Acquiring status of the reader,
- Capturing fingerprints,
- Starting and stopping image stream, and
- Resetting and calibrating the reader.

To acquire a reader, first instantiate a `XReaderCollection` object and call `XReaderCollection.GetReaders()` which returns the attached readers. Iterate through this list to look at the description object for the desired reader. Once finished with all of the `XReader` objects in the `XReaderCollection`, ensure that you call the `Dispose()` method to destroy the `XReaderCollection` object and release system resources.

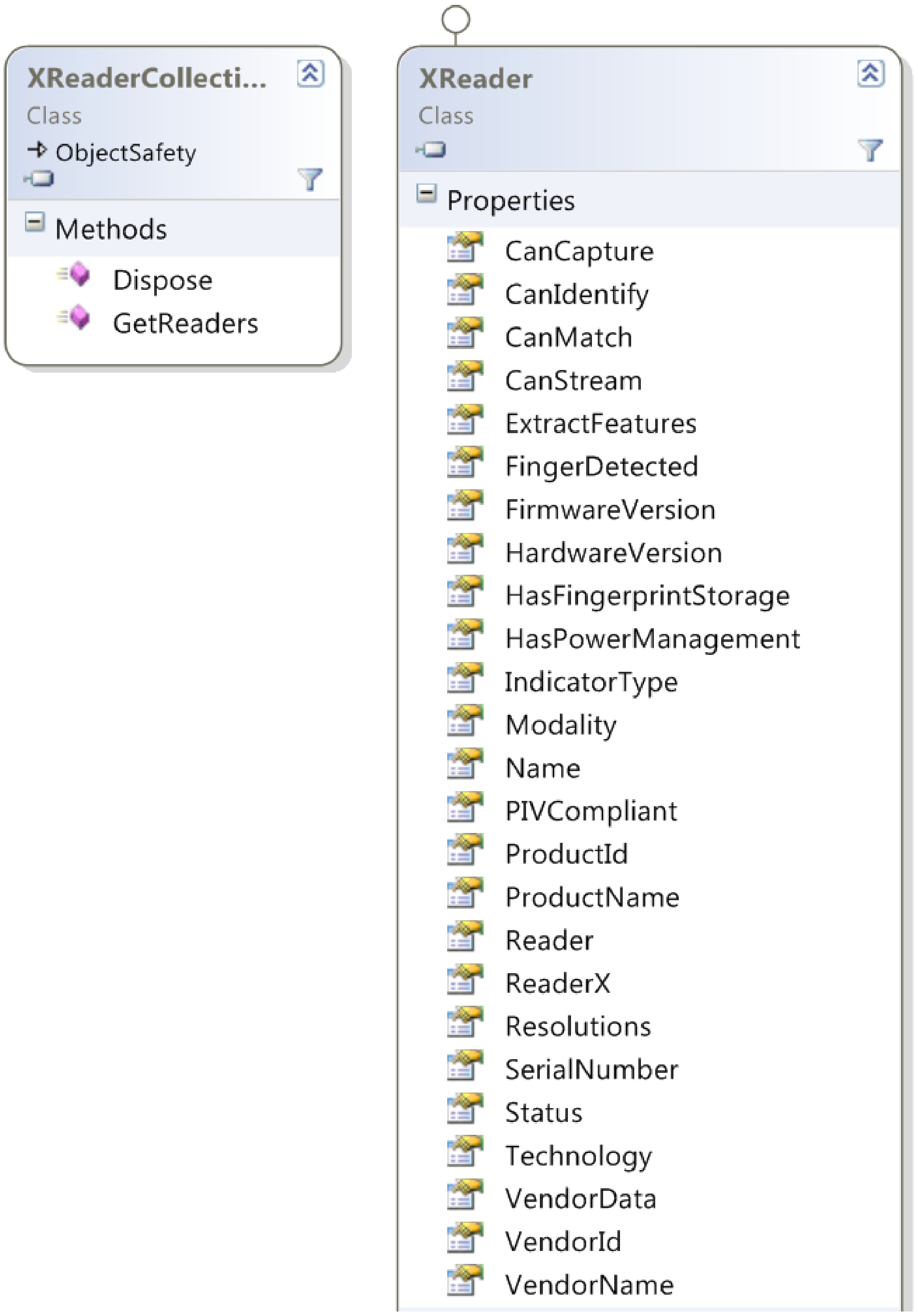
A Note About Internet Explorer and Process Merging

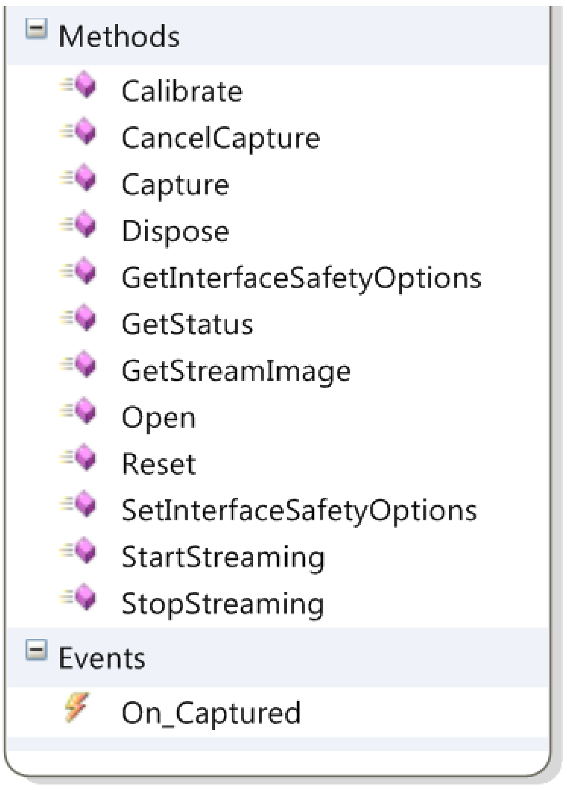
By default, Internet Explorer merges browser processes where it can, beginning with IE8. If your application is open in more than one browser window, users may experience erroneous behavior when your application calls `XReader.Dispose()` from one window.

To prevent this, users can use the **File > New Session** command to open a new window that does not merge processes with existing windows. You can also prevent IE from merging processes by using the **-nomerge** command line option when launching IE OR change the registry setting of `HKEY_CURRENT _USER\Software\Microsoft\Internet Explorer\Main` to set `SessionMerging` to 0. Note however that these options will affect IE's efficiency and startup time.

Class Diagrams

The UML diagrams below show the `XReaderCollection` and `XReader` classes.


XReaderCollecti...
Class
ObjectSafety
Methods
Dispose
GetReaders
XReader
Class
Properties
CanCapture
CanIdentify
CanMatch
CanStream
ExtractFeatures
FingerDetected
FirmwareVersion
HardwareVersion
HasFingerprintStorage
HasPowerManagement
IndicatorType
Modality
Name
PIVCompliant
ProductId
ProductName
Reader
ReaderX
Resolutions
SerialNumber
Status
Technology
VendorData
VendorId
VendorName



The main reader hardware management methods in the `XReader` class are:

Table 22. Hardware management methods

Function	Description
Open	Open a device and return the device capabilities. This method establishes an exclusive link to the device; no other processes will be able to use the device until you close it. The application *must* open the device before use.
GetStatus	Get the status for a device. You would normally check the device status before captures to ensure that the device is functioning and there are no error conditions. Returns an object of type `ReaderStatus` which describes the current status of the reader.
Calibrate	Calibrate a device. Some devices are self-calibrating. Ambient light or temperature can affect calibration, for some devices. Calibration can take several seconds.

Table 22. Hardware management methods

Function	Description
Reset	Do a hardware reset on the reader. Hardware resets are typically needed only after a hardware problem (e.g., the device is unplugged or receives an electrostatic shock). Hardware resets typically only take a few milliseconds.
Dispose	Close device, release memory, remove child objects.

Capturing Fingerprints

The `Capture` function of the `XReader` class captures a fingerprint image for

- Enrollment (as part of the process described on page *16)*
- Identifying users with `Identify`
- Verifying a specific user identity with `Compare`

The primary fingerprint capture methods are:

Table 23. Fingerprint capture functions

Function	Description
Capture	Captures a fingerprint image from the open reader. This function signals the reader that a fingerprint is expected, and blocks until a fingerprint is received, capture fails or the reader times out.
CancelCapture	Cancel a pending capture

Streaming Fingerprints

Not all readers support streaming mode. To determine if a specific reader supports this feature, check the value of the `XReader` property `CanStream`.

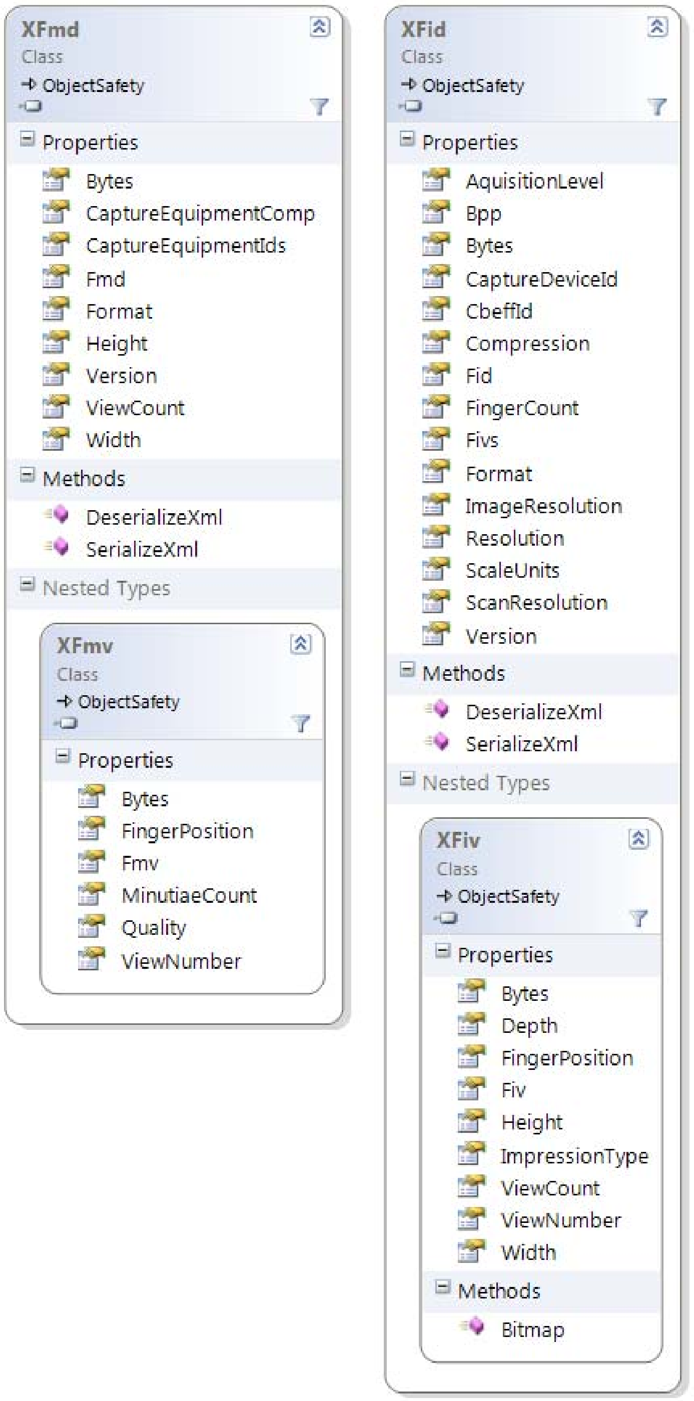
The streaming methods in `XReader` are:

Table 24. Streaming Functions

Function	Description
StartStreaming	Put the reader into streaming mode. In this mode, the application must call `GetStreamImage()` to acquire images from the stream.
GetStreamImage	Capture a fingerprint image from the streaming data. After this function returns, the reader remains in streaming mode. Frame selection, scoring and other image processing are not performed by this function.
StopStreaming	End streaming mode

Managing Fingerprint Data

The ActiveX API implements fingerprint data as shown in the UML diagram below:



In addition to the usual methods for working with data, this class includes the methods below for serializing and deserializing (as described in *Importing the U.are.U .NET package* on *page 44*.

Table 25. Serializing and Deserializing

Function	**Description**
SerializeXml	Static method which serializes FMDs and FIDs into an XML string. Note that raw data is encoded as base-64.
DeserializeXml	Static method which de-serializes XML-encoded FMDs and FIDs back into a U.are.U FMD or FID data structure.

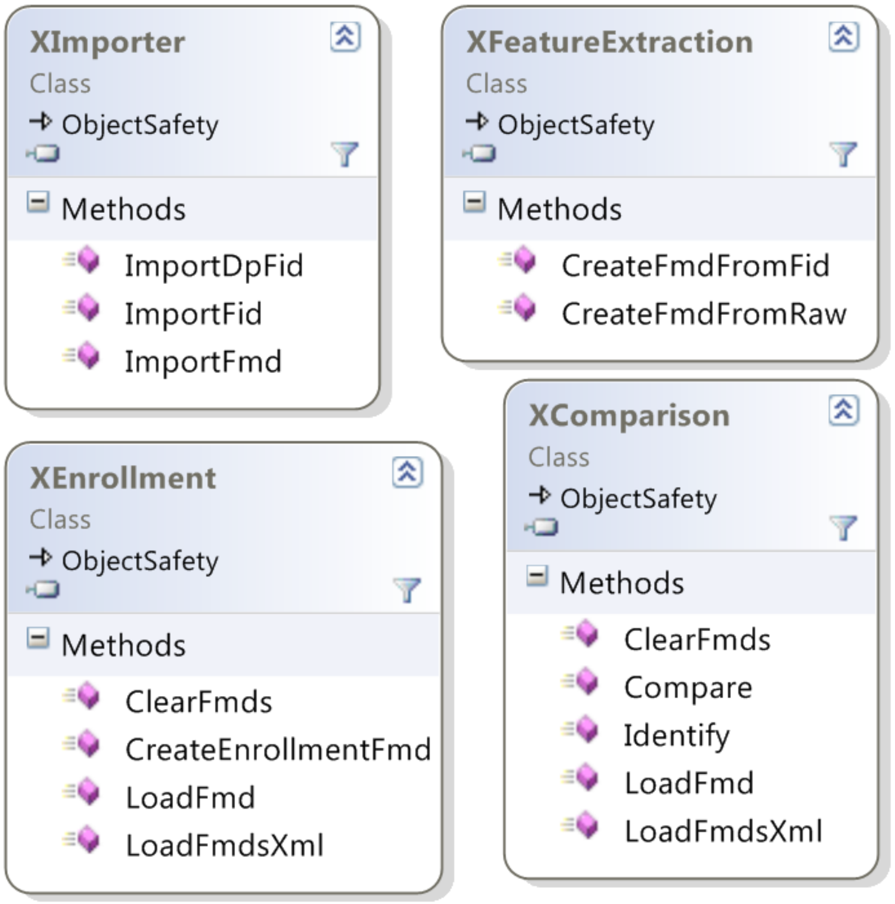
Accessing the FingerJet Engine

The FingerJet Engine interfaces provide functionality to

- Extract fingerprint features (create FMDs),
- Identify and compare FMDs, and
- Create enrollment FMDs.

Refer to *Understanding the Data Flow* on *page 15* for a description of enrollment and comparison terminology and data flow.

The UML diagram below shows the relevant classes:



Creating FMDs from images

The `CreateFmdFromFid()` method can be used in two ways:

1. Extract fingerprint minutiae from a raw image and create a FMD.
2. Extract fingerprint minutiae from a FID and create a FMD.

The following limitations are applied to the raw images and FIDs:

- 8 bits per pixel
- no padding
- square pixels (horizontal and vertical dpi are the same)

The size of the resulting FMD will vary depending on the minutiae in a specific fingerprint.

Identification and Comparison

`Identify()` identifies a single FMD against an array of FMDs. This function takes as inputs:

- A single view in an FMD
- An array of FMDs (each FMD can contain up to 16 views) to compare
- The desired number of candidates to return
- The threshold for False Positive Identification Rate (FPIR) that is permitted

and returns matches as an array of integer pairs which provide the finger index and view index of each match.

Each time a view has a score lower than the threshold FPIR, that view is marked as a possible candidate. Then when all possible candidates are identified (i.e., they meet the threshold), they are ranked by their score. Finally, the function returns as many candidates as requested, based on the candidates with the lowest dissimilarity score. For a discussion of setting the threshold as well as the statistical validity of the dissimilarity score and error rates, consult *NIST Fingerprint Image Quality (NFIQ)* on *page 19*.

`Compare()` takes two single views from two FMDs and returns a dissimilarity score indicating the quality of the match.

The majority of applications should use the `Identify` method to implement both identification and verification. However, in a few special cases, e.g., using multi-modal biometrics, or doing statistical risk assessment, the `Compare` method allows you to compare two FMVs to determine their actual degree of dissimilarity. This is useful for accuracy testing and diagnostics and is not intended to be used in final applications for actual fingerprint recognition. The `Compare` method returns a ***dissimilarity score*** with values:

- 0 = fingerprints are NOT dissimilar (i.e., they MATCH perfectly).

- `maxint` (#7FFFFFFF or 2147483647) = fingerprints are completely dissimilar (i.e., DO NOT match).
- Values close to 0 indicate very close matches, values closer to `maxint` indicate very poor matches.

The table below shows the relationship between the scores returned from `Compare` and the false match error rates observed in our test. The dissimilarity score distribution is estimated based on our internal testing, and may not be representative of the actual rate that will be observed in deployment.

Dissimilarity Score	False Match Rate
2147483	.1%
214748	.01%
21474	.001%
2147	.0001%

Table 26. Methods for Verifying and Identifying Fingerprints

Function	Description
Compare	Compare two FMDs; supported formats are: Gold SDK, One Touch SDK, ANSI and ISO.
Identify	Identify a FMD: given an array of FMDs, this function returns an array of candidates that match the original fingerprint (a FMV within a FMD) within the threshold of error. Supported formats are: Gold SDK, One Touch SDK, ANSI and ISO.

Enrollment

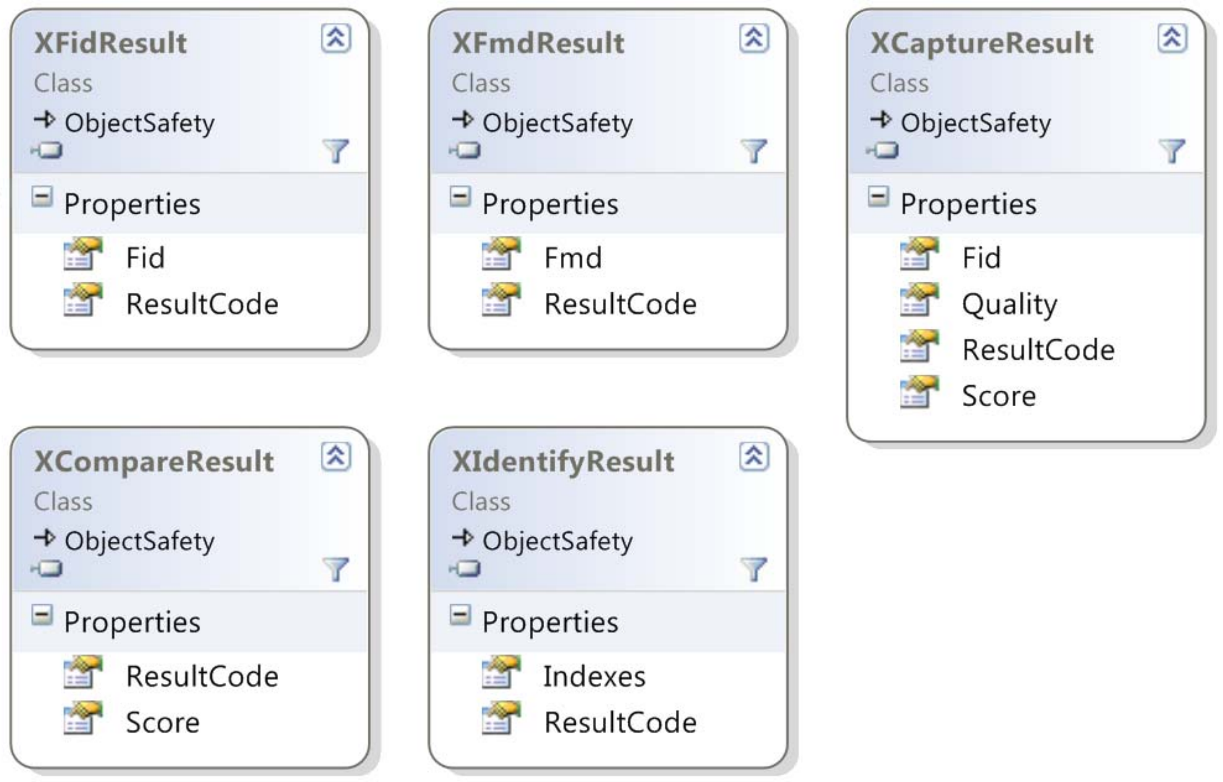
`CreateEnrollmentFmd()` creates and returns an enrollment FMD. The method takes as input:

- Which format to use, e.g., ANSI or ISO
- An `ArrayList` object containing the FMDs to enroll

The recommended sequence in which to enroll FMDs is to send `CreateEnrollmentFmd()` four captured FMDs in an array. If four FMDs is not enough, you can capture a FID, convert it to an additional FMD, add it to the array and call `CreateEnrollmentFMD()` again. If `CreateEnrollmentFMD()`, fails again, you can continue to add FMDs to the array and call `CreateEnrollmentFMD()` until an enrollment FMD is created successfully.

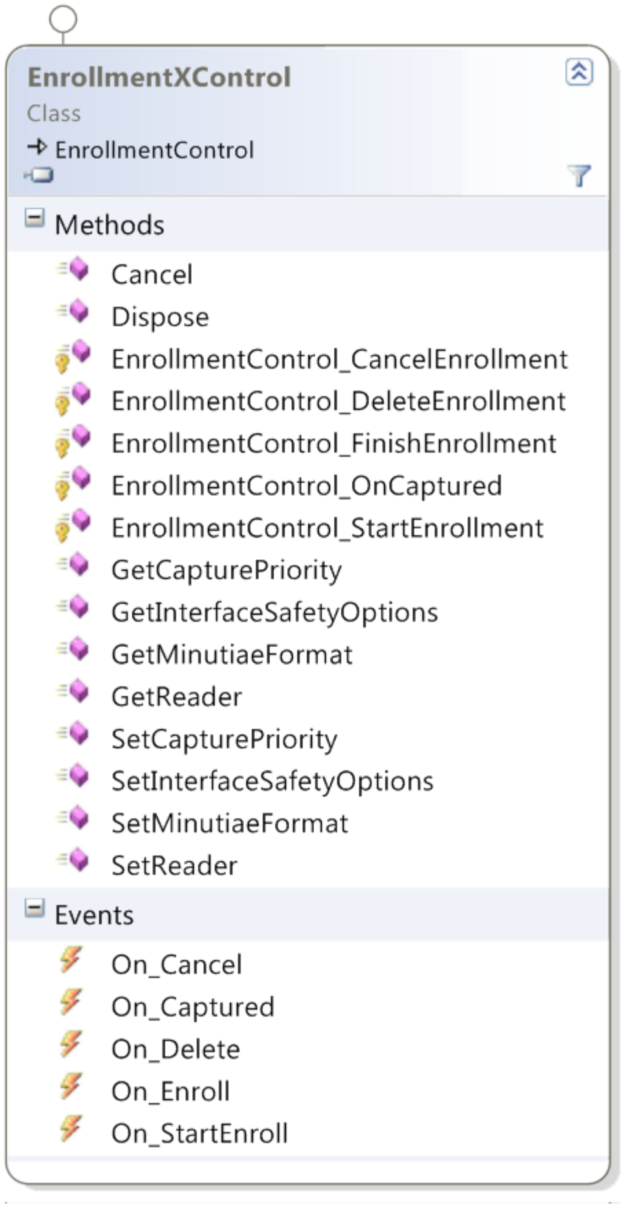
Common Data Structures for Results

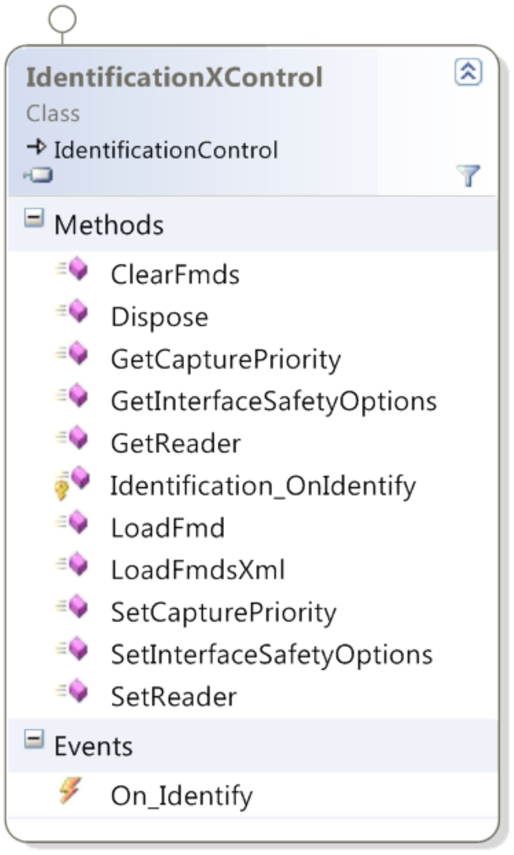
The following classes are used when data is returned as a result of a U.are.U ActiveX operation.



Pre-Built Controls for Enrollment and Identification

In addition to the basic API, the SDK contains two pre-built controls that allow you to quickly add enrollment or identification into your application. These controls provide the same functionality as the DigitalPersona One Touch products.


EnrollmentXControl
Class
EnrollmentControl
Methods
Cancel
Dispose
EnrollmentControl_CancelEnrollment
EnrollmentControl_DeleteEnrollment
EnrollmentControl_FinishEnrollment
EnrollmentControl_OnCaptured
EnrollmentControl_StartEnrollment
GetCapturePriority
GetInterfaceSafetyOptions
GetMinutiaeFormat
GetReader
SetCapturePriority
SetInterfaceSafetyOptions
SetMinutiaeFormat
SetReader
Events
On_Cancel
On_Captured
On_Delete
On_Enroll
On_StartEnroll



JavaPOS 11

The U.are.U JavaPOS API is built on the U.are.U Java API. This JavaPOS API is geared for Point of Sale applications and has the following features:

- **Fully compliant with JavaPOS 1.13.** The U.are.U JavaPOS API conforms to the specifications for the Biometrics device category in Chapter 5, "Biometrics," of the *UnifiedPOS Retail Peripheral Architecture*, Version 1.13 (July 15, 2009). The complete UPOS documentation is available at *http://www.nrf-arts.org/UnifiedPOS/default.htm*.
- **Backward compatible with DigitalPersona's previous JavaPOS product (U.are.U UPOS for JavaPOS SDK),** with only a few caveats. This new API is the result of merging the previous JavaPOS SDK with the U.are.U SDK - the internal architecture has been completely rewritten. For users of the previous SDK, the updated SDK means an upgraded internal architecture for more robust performance -- up to ten times faster for identification. If you are upgrading an existing application, see *Upgrading from U.are.U UPOS for OPOS/JavaPOS* on *page 26* for more details.
- **In addition to the JavaPOS API, the U.are.U SDK includes a JavaPOS Device Service object**, which can be used with any JavaPOS Device Control for the Biometrics device category.
- **Because the JavaPOS API is built as an adjunct to the Java API, applications can use both the JavaPOS standard operations AND use the U.are.U Java API** (described in *The Java API* on *page 37*). Java methods can be used to access streaming features or to import data.

Terminology Note

The U.are.U JavaPOS API conforms to the standard terminology used by the JavaPOS spec from 2009 whereas the other APIs in the U.are.U SDK generally use terminology that matches evolving industry standards. If you are going to use the U.are.U Java API along with JavaPOS, you need to note that in JavaPOS, the extracted fingerprint data (template) is stored in a biometric information record (BIR). The templates created through enrollment and capture are equivalent to Fingerprint Minutiae Data (FMD) records in the other APIs of the U.are.U SDK. Note that JavaPOS templates include a 10- or 45-byte JavaPOS header in addition to the data itself.

Working with Fingerprint Data in JavaPOS

The U.are.U JavaPOS API provides two types of fingerprint data: raw images and fingerprint templates. The data flow for JavaPOS applications is the same as described in Chapter 3, *Understanding the Data Flow,* on *page 15* except that JavaPOS data has additional JavaPOS headers and JavaPOS does not use the FID/FMD terminology used by U.are.U. Note that there is no easy method provided in the API for converting data from JavaPOS BIRs to FMDs.

Fingerprint Data for Raw Images (Captures)

Per the JavaPOS spec, raw fingerprint scans (images) are available in the **RawSensorData** property when **StatusUpdateEvent** triggers.

Fingerprint Data for Captures and Enrollment Templates (BIRs)

The U.are.U JavaPOS API now supports four different JavaPOS biometric information record (BIR) formats, as set by the *Algorithm* property:

Value	Meaning
0	Default (Same as 1)
1	DigitalPersona format with 45-byte header that conforms to JavaPOS 1.13 spec. New captures and enrollments will be done in DigitalPersona format with 45-byte headers. Verify and identify will correctly compare with BIRs that have 45-byte headers, the previous 10-byte headers or a mixture.
2	ANSI INSITS 378-2004. New captures and enrollments will be created in ANSI format with a 45-byte JavaPOS header. Verify and identify will require that all BIRs be in ANSI format.
3	ISO/IEC 19794-2:2005. New captures and enrollments will be created in ISO format with a 45-byte JavaPOS header. Verify and identify will require that all BIRs be in ISO format.
4	JavaPOS 1.x compatible. This is the previous DigitalPersona format with 10-byte header. This format was the default (and only) format used in the U.are.U UPOS for JavaPOS SDK. New captures and enrollments will be created in DigitalPersona format with 10-byte headers. Verify and identify will correctly compare with BIRs that have 10-byte headers.

Identification and verification require that all BIRs be in the same format. Stored enrollment BIRs can be in four formats:

1. DigitalPersona format (10-byte headers, 45-byte headers or a mixture),
2. ANSI (with 45-byte headers)
3. ISO (with 45-byte headers)
4. JavaPOS 1.x compatible (DigitalPersona format with 10-byte headers).

For example, if the *Algorithm* property is set to ANSI and you receive ANSI captures, those fingerprints cannot be matched against templates that are stored in a DigitalPersona format.

The value of the *Algorithm* property must be set before the device is enabled (i.e., after the device is opened and claimed, but before it is enabled).

Working with DigitalPersona Record Formats

The format for the new default 45-byte headers is provided in the JavaPOS spec. The first 10 bytes of the 45-byte header are the same as the previous 10-byte header except for the header version number.

In both 10-byte and 45-byte headers, the 5th byte specifies the header format:

- The 5th byte is 0x10 for a 10-byte header
- The 5th byte is 0x20 for a 45-byte header

Converting vs. Re-Enrolling

DigitalPersona recommends that you re-enroll users rather than convert your existing fingerprint data to a newer format.

While it is possible to convert old templates to newer formats, the conversion process may occasionally result in templates that are not recognized. We therefore recommend that if you wish to convert to a newer format, that you simply re-enroll users with the new format.

Getting Device-Specific Information with DirectIOEvent

The **DirectIOEvent** event is fired by a Service Object to deliver vendor-specific events to the application. The U.are.U JavaPOS API uses DirectIOEvent events to notify the application about device connection and disconnection and intermediate events such as finger touch or removal.

Syntax

upos::events::DirectIOEvent
EventNumber: int32 { read-only }
Data: int32 { read-write }
Obj: object { read-write }

Properties

This event contains the following attributes:

Property	Type	Description
EventNumber	*int*	Constants whose specific values are assigned by the Service Object
Data	*int*	Not used
Obj	*object*	Name of the fingerprint reader

EventNumber Return Values

These constants are defined in the `com.digitalpersona.javapos.services.biometrics.dpfpconstants` class.

EventNumber	Description
DP_EVENT_DISCONNECT	The fingerprint reader was disconnected.

DP_EVENT_RECONNECT	The fingerprint reader was reconnected.
DP_EVENT_FINGER_TOUCHED	The fingerprint reader was touched.
DP_EVENT_FINGER_GONE	The finger was removed from the fingerprint reader.
DP_EVENT_COMPLETED	Capture completed.
DP_EVENT_IMAGE_READY	Raw image (from a capture) is available.
DP_EVENT_SAMPLE_QUALITY	Information about quality of the sample is available.

Implementation Notes

The following table provides information about how UPOS and JavaPOS properties, methods, and events are implemented in the U.are.U JavaPOS API.

Name	Implemented	Notes about Implementation
UPOS Common Properties		
AutoDisable	No	This property is initialized to **false** in the **open** method. Changing its value will have no effect.
CapCompareFirmwareVersion	Yes	This property is initialized to **false** in the **open** method since firmware comparison is not supported.
CapPowerReporting	Yes	This property is initialized to **JPOS_PR_NONE** in the **open** method since power reporting is not supported.
CapStatisticsReporting	Yes	This property is initialized to **false** in the **open** method since statistics reporting is not supported.
CapUpdateFirmware	Yes	This property is initialized to **false** in the **open** method since firmware updating is not supported.
CapUpdateStatistics	Yes	This property is initialized to **false** in the **open** method since statistics updating is not supported.
CheckHealthText	Yes	This property is initialized to an **empty string** in the **open** method. It is never modified since device health checking is not supported.
Claimed	Yes	
DataCount	Yes	
DataEventEnabled	Yes	Has value **true** when an enroll or capture or verify capture is in progress.

Name	Implemented	Notes about Implementation
DeviceEnabled	Yes	Has value **true** when when the service will return device events. The Algorithm property cannot be changed when the device is enabled.
DeviceServiceDescription	Yes	This property is set to "U.are.U Biometrics Device Service for JavaPOS, (c) 2007-2012 DigitalPersona, Inc."
DeviceServiceVersion	Yes	This property is set to 1013000.
FreezeEvents	Yes	
PhysicalDeviceDescription	Yes	This property contains the product name, serial number, and vendor name for the DigitalPersona device.
PhysicalDeviceName	Yes	This property contains the product name of the DigitalPersona device.
PowerNotify	Yes	This property is initialized to **JPOS_PN_DISABLED** in the **open** method since power reporting is not supported.
PowerState	Yes	This property is initialized to **JPOS_PS_UNKNOWN** in the **open** method since power reporting is not supported.
State	Yes	
Specific Properties		
Algorithm	Yes	Specifies the data format used. See *Fingerprint Data for Captures and Enrollment Templates (BIRs)* on *page 26* for details.
AlgorithmList	Yes	See *Fingerprint Data for Captures and Enrollment Templates (BIRs)* on *page 26* for details.
BIR	Yes	
CapPrematchData	Yes	This property is initialized to **false** in the **open** method since pre-matching is not supported.
CapRawSensorData	Yes	
CapRealTimeData	Yes	This property is initialized to **false** in the **open** method.
CapSensorColor	Yes	This property is initialized to **BIO_CSC_GRAYSCALE** in the **open** method.
CapSensorOrientation	Yes	This property is initialized to **BIO_CSO_NORMAL** in the **open** method.
CapSensorType	Yes	This property is initialized to **BIO_CST_FINGERPRINT** in the **open** method.

Name	Implemented	Notes about Implementation
CapTemplateAdaptation	Yes	This property is initialized to **false** in the **open** method since template adaptation is not supported.
RawSensorData	Yes	Unkown OR contains raw sensor data. The data in this property exists when **CapRawSensorData** is set to **true**. and **StatusUpdateEvent** triggers.
RealTimeDataEnabled	Yes	This property is initialized to **false** in the **open**.
SensorBPP	Yes	This property is initialized to **8** in the **open** method. No other value is supported.
SensorColor	Yes	This property is initialized to **BIO_CSC_GRAYSCALE** in the **open** method. No other value is supported.
SensorHeight	Yes	This property is initialized to **0** in the **open** method. No other value is supported.
SensorOrientation	Yes	This property is initialized to **BIO_SO_NORMAL** in the **open** method. No other value is supported.
SensorType	Yes	This property is initialized to **BIO_ST_FINGERPRINT** in the **open** method.
SensorWidth	Yes	This property is initialized to **0** in the **open** method. No other value is supported.
UPOS Common Methods		
Open	Yes	
Close	Yes	
ClaimDevice	Yes	
ReleaseDevice	Yes	
CheckHealth	Yes	Only internal health check is supported. Attempts to perform an external or interactive check cause an exception to be thrown.
ClearInput	Yes	
ClearInputProperties	Yes	
ClearOutput	No	This method is not supported by the Biometrics device category.
DirectIO	Partial	An exception is thrown since device direct I/O is not supported.
CompareFirmwareVersion	Yes	An exception is thrown since **CapCompareFirmwareVersion** is set to false.

Name	Implemented	Notes about Implementation
ResetStatistics	Yes	An exception is thrown since **CapUpdateStatistics** is set to false.
RetrieveStatistics	Yes	An exception is thrown since **CapStatisticsReporting** is set to false.
UpdateFirmware	Yes	An exception is thrown since **CapUpdateFirmware** is set to false.
UpdateStatistics	Yes	An exception is thrown since **CapUpdateStatistics** is set to false.
Specific Methods		
beginEnrollCapture	Yes	
beginVerifyCapture	Yes	
endCapture	Yes	
identify	Yes	
identifyMatch	Yes	
processPrematchData	Yes	An exception is thrown since **CapPrematchData** is set to false.
verify	Yes	
verifyMatch	Yes	
UPOS Events		
DataEvent	Yes	The following event types are supported: BIO_DATA_ENROLL - Enrollment template created BIO_DATA_VERIFY - Verification template created
DirectIOEvent	Yes	See *Getting Device-Specific Information with DirectIOEvent* on *page 28* for details.
ErrorEvent	Yes	**resultCode** - Standard JavaPOS Result Code **resultCodeExtended** - DigitalPersona Result Code - see *Errors and Exceptions* on *page 33* for details **errorLocus** - EL_INPUT **errorResponse** - ER_CLEAR

Name	Implemented	Notes about Implementation
OutputCompleteEvent	No	This event is not supported by the Biometrics device category.
StatusUpdateEvent	Yes	Only occurs when **RawSensorData** has received a raw image from a capture.

Exceptions

The files `jpos.Const.*` contain the JavaPOS standard exception codes. The following exceptions are thrown by the U.are.U JavaPOS API:

- JPOS_E_FAILURE
- JPOS_E_ILLEGAL
- JPOS_E_NOHARDWARE
- JPOS_E_TIMEOUT

Device-Related Error Codes

For the device-related result codes returned by *EventNumber* after the **DirectIOEvent** event, see *EventNumber Return Values* on *page 28*.

The following error codes are returned in the *ResultCodeExtended* property of the **ErrorEvent** object.

IMPORTANT: You should always use the names of the return codes, such as FT_OK, rather than the numeric values because these values may change at any time, without notice. The numbers are provided as a reference for the error codes in the sample application.

Value	Result Code	Description
-1	FT_ERR_NO_INIT	The fingerprint feature extraction module or the fingerprint comparison module is not initialized.
-2	FT_ERR_INVALID_PARAM	One or more parameters are not valid.
-4	FT_ERR_IO	A generic I/O file error occurred.
-7	FT_ERR_NO_MEMORY	There is not enough memory to perform the action.
-8	FT_ERR_INTERNAL	An unknown internal error occurred.
-10	FT_ERR_UNKNOWN_DEVICE	The requested device is not known.
-11	FT_ERR_INVALID_BUFFER	A buffer is not valid.
-33	FT_ERR_UNKNOWN_EXCEPTION	An unknown exception occurred.

OPOS 12

The U.are.U OPOS API is built on the U.are.U C/C++ API. The OPOS API is geared for Point of Sale applications and has the following features:

- **Fully compliant with OPOS 1.13.** The U.are.U OPOS API conforms to the specifications for the Biometrics device category in Chapter 5, "Biometrics," of the *UnifiedPOS Retail Peripheral Architecture*, Version 1.13 (July 15, 2009). The complete UPOS documentation is available at *http://www.nrf-arts.org/UnifiedPOS/default.htm*.
- **Backward compatible with DigitalPersona's previous OPOS product (U.are.U UPOS for OPOS SDK),** with a few minor caveats. This new API is the result of merging the previous OPOS SDK with the U.are.U SDK. For users of the previous SDK, the updated SDK means an upgraded internal architecture for more robust performance -- up to ten times faster for identification. If you are upgrading an existing application, see *Upgrading from U.are.U UPOS for OPOS/JavaPOS* on *page 26* for more details.
- **The OPOS API is a set of COM objects** that acts as an extension to the DigitalPersona fingerprint reader driver, providing an OPOS-compliant application interface to DigitalPersona products. It consists of a Control Object (CO) (an ActiveX® Control) for the OPOS Biometrics device category and a Service Object (SO).
- **Because the OPOS API is built as an adjunct to the C/C++ API, applications can use both the OPOS standard operations AND use the U.are.U C/C++ API** (described in *The C/C++ APIs* on *page 29*). C/C++ methods can be used to access streaming features or to import data.

Terminology Note

The U.are.U OPOS API conforms to the standard terminology used by the OPOS spec from 2009 whereas the other APIs in the U.are.U SDK generally use terminology that matches evolving industry standards. If you are going to use the U.are.U C/C++ API along with OPOS, you need to note that in OPOS, the extracted fingerprint data (template) is stored in a biometric information record (BIR). The templates created through enrollment and capture are equivalent to Fingerprint Minutiae Data (FMD) records in the other APIs of the U.are.U SDK. Note that OPOS templates include a 12- or 45-byte OPOS header in addition to the data itself.

Working with Fingerprint Data in OPOS

The U.are.U OPOS API provides two types of fingerprint data: raw images and fingerprint templates. The data flow for OPOS applications is the same as described in Chapter 3, *Understanding the Data Flow,* on *page 15* except that OPOS data has additional OPOS headers and OPOS does not use the FID/FMD terminology used by U.are.U. Note that there is no easy method provided in the API for converting data from OPOS BIRs to FMDs.

Fingerprint Data for Raw Images (Captures)

Per the OPOS spec, raw fingerprint scans (images) are available in the **RawSensorData** property when **StatusUpdateEvent** triggers.

Fingerprint Data for Captures and Enrollment Templates (BIRs)

The U.are.U OPOS API now supports four different OPOS biometric information record (BIR) formats, as set by the *Algorithm* property:

Value	Meaning
0	Default (Same as 1)
1	DigitalPersona format with 45-byte header that conforms to OPOS 1.13 spec. New captures and enrollments will be done in DigitalPersona format with 45-byte headers. Verify and identify will correctly compare with BIRs that have 45-byte headers, the previous 12-byte headers or a mixture.
2	ANSI INSITS 378-2004. New captures and enrollments will be created in ANSI format with a 45-byte OPOS header. Verify and identify will require that all BIRs be in ANSI format.
3	ISO/IEC 19794-2:2005. New captures and enrollments will be created in ISO format with a 45-byte OPOS header. Verify and identify will require that all BIRs be in ISO format.
4	OPOS 1.x compatible. This is the previous DigitalPersona format with 12-byte header. This format was the default (and only) format used in the U.are.U UPOS for OPOS SDK. New captures and enrollments will be created in DigitalPersona format with 12-byte headers. Verify and identify will correctly compare with BIRs that have 12-byte headers.

Identification and verification require that all BIRs be in the same format. The BIRs can be in four formats:

1. DigitalPersona format (12-byte headers, 45-byte headers or a mixture),
2. ANSI (with 45-byte headers)
3. ISO (with 45-byte headers)
4. OPOS 1.x compatible (DigitalPersona format with 12-byte headers).

For example, if the *Algorithm* property is set to ANSI and you receive ANSI captures, those fingerprints cannot be matched against templates that are stored in a DigitalPersona format.

The value of the *Algorithm* property must be set before the device is enabled (i.e., after the device is opened and claimed, but before it is enabled).

Working with DigitalPersona Record Formats

The format for the new default 45-byte headers is provided in the OPOS spec. The first 12 bytes of the 45-byte header are the same as the previous 12-byte header except for the header version number.

In both 12-byte and 45-byte headers, the 5th byte specifies the header format:

- The 5th byte is 0x10 for a 12-byte header

- The 5th byte is 0x20 for a 45-byte header

Converting vs. Re-Enrolling

DigitalPersona recommends that you re-enroll users rather than convert your existing fingerprint data to a newer format.

While it is possible to convert old templates to newer formats, the conversion process may occasionally result in templates that are not recognized. We therefore recommend that if you wish to convert to a newer format, that you simply re-enroll users with the new format.

Getting Device-Specific Information with DirectIOEvent

The **DirectIOEvent** event is fired by a Service Object to deliver vendor-specific events to the application. The U.are.U JavaPOS API uses DirectIOEvent events to notify the application about device connection and disconnection and intermediate events such as finger touch or removal.

This chapter contains specific information about the U.are.U SDK implementation of the OPOS Control.

DataEvent

Syntax

```
<< event >>  upos::events::DataEvent
                    Status:  int32  { read-only }
```

Description

This event is fired to provide input data from the fingerprint reader to the application. The actual input data is placed in one or more device-specific properties.

Attribute

This event contains the following attribute:

Attribute	Type	Description
Status	*int32*	BIO_DATA_ENROLL if enroll capture is completed. BIO_DATA_VERIFY if verify capture is completed.

DirectIOEvent

Syntax

```
<< event >>  upos::events::DirectIOEvent
                       EventNumber:  int32  { read-only }
                       Data:  int32  { read-write }
                       Obj:  object  { read-write }
```

Description

This event is fired by the Service Object (SO) to communicate directly with the application. **DirectIOEvent** is used in the U.are.U SDK to notify the user of the image-capturing status, fingerprint reader connection status, and image quality, etc., whenever required.

Attributes

This event contains the following attributes:

Attribute	Type	Description
EventNumber	*int32*	Event number whose specific values are assigned by the SO.
Data	*int32*	Additional numeric data. Specific values vary by the *EventNumber* and the SO.
Obj	*object*	Additional data whose use varies by the *EventNumber* and the SO.

EventNumber Return Values

EventNumber	Description
1	The fingerprint reader was disconnected.
2	The fingerprint reader was reconnected.
3	The fingerprint reader was touched.
4	The finger was removed from the fingerprint reader.
5	A fingerprint image is ready for processing.
6	Provides information about the quality of the fingerprint image.
7	A supplied fingerprint credential was added to the fingerprint enrollment template.
8	The fingerprint capture operation was stopped.

Data Return Values

EventNumber[1]	**Data**	**Description**
6	DP_QUALITY_GOOD, DP_QUALITY_NONE, etc.	Can be any value from the DP_SAMPLE_QUALITY enumeration in the table on the next page.
7	1, 2, 3, etc.	Holds the number of images added.

1. For every other *EventNumber* (1 through 5 and 8), *Data* holds the value 0 (Not Supported).

DP_SAMPLE_QUALITY Enumeration

Value	Meaning
DP_QUALITY_GOOD (0)	The image is of good quality.
DP_QUALITY_NONE (1)	There is no image.
DP_QUALITY_TOOLIGHT (2)	The image is too light.
DP_QUALITY_TOODARK (3)	The image is too dark.
DP_QUALITY_TOONOISY (4)	The image is too noisy.
DP_QUALITY_LOWCONTR (5)	The image contrast is too low.
DP_QUALITY_FTRNOTENOUGH (6)	The image does not contain enough information.
DP_QUALITY_NOCENTRAL (7)	The image is not centered.

Obj Return Values

EventNumber	Object
1	The fingerprint reader was disconnected.
2	The fingerprint reader was reconnected.
3	The fingerprint reader was touched.
4	The finger was removed from the fingerprint reader.
5	NULL (Not Supported).
6	Provides information about the quality of the fingerprint image.
7	A supplied fingerprint credential was added to the fingerprint enrollment template.
8	The fingerprint capture operation was stopped.

StatusUpdateEvent

Syntax

<< event >> upos::events::StatusUpdateEvent
Status: int32 { read-only }

Description

This event is used in the U.are.U SDK to notify the user that a raw image is available for use.

Attribute

This event contains the following attribute:

Attribute	Type	Description
Status	*int32*	The *Status* parameter notifies the user that raw image data is available.

Status Return Values

StatusUpdateEvent	Value	Meaning
1	BIO_SUE_RAW_DATA	Raw image data is available.

Implementation Notes

The following table provides information about how UPOS and JavaPOS properties, methods, and events are implemented in the U.are.U JavaPOS API.

The following table provides information about how UPOS properties, methods, and events are implemented in the DigitalPersona U.are.U SDK.

Name	Implemented[1]	Comment
UPOS Common Properties		
AutoDisable	Partial	This property is initialized to false in the **open** method. It is not modified during execution of the application since it is not required to automatically disable the device when data is received.

Name	Implemented[1]	Comment
BinaryConversion	Partial	This property is initialized to OPOS_BC_NONE in the **open** method and is not modified during execution of the application.
CapCompareFirmwareVersion	Partial	This property is initialized to false in the **open** method. It is not modified during execution of the application since firmware version comparison is not supported.
CapPowerReporting	Partial	This property is initialized to OPOS_PR_NONE in the **open** method. It is not modified during execution of the application since power reporting is not supported.
CapStatisticsReporting	Partial	This property is initialized to false in the **open** method. It is not modified during execution of the application since statistics reporting is not supported.
CapUpdateFirmware	Partial	This property is initialized to false in the **open** method. It is not modified during execution of the application since firmware updating is not supported.
CapUpdateStatistics	Partial	This property is initialized to false in the **open** method. It is not modified during execution of the application since statistics updating is not supported.
CheckHealthText	No	
Claimed	Yes	This property is initialized to false in the **open** method. It is set to true on claim and set to false again on release.
ControlObjectDescription	Yes	
ControlObjectVersion	Yes	
DataCount	Partial	This property is initialized to 0 in the **open** method and is not modified during execution of the application.
DataEventEnabled	Partial	This property is initialized to false in the **open** method and is set to true after a successful claim operation.
DeviceDescription	Yes	
DeviceEnabled	Partial	This property is initialized to false in the **open** method and is set to true after a successful claim operation. Since the **AutoDisable** property is never set to true, **DeviceEnabled** always remains true.
DeviceName	Yes	
FreezeEvents	Partial	This property is initialized to false in the **open** method. It is not modified during execution of the application since it is not required to freeze events.

Name	**Implemented**[1]	**Comment**
OpenResult	Yes	
OutputID	No	This property is not supported by the Biometrics device category.
PowerNotify	Partial	This property is initialized to OPOS_PN_DISABLED in the **open** method. It is not modified during execution of the application since power reporting is not supported.
PowerState	Partial	This property is initialized to OPOS_PS_UNKNOWN in the **open** method. It is not modified during execution of the application since power reporting is not supported.
ResultCode	Yes	
ResultCodeExtended	No	
ServiceObjectDescription	Yes	
ServiceObjectVersion	Yes	
State	Partial	This property is initialized to OPOS_S_IDLE in the **open** method and is set to OPOS_S_CLOSED in the **close** method.
Specific Properties		
Algorithm	Yes	See *Fingerprint Data for Captures and Enrollment Templates (BIRs)* on *page 85* for details.
AlgorithmList	Yes	See *Fingerprint Data for Captures and Enrollment Templates (BIRs)* on *page 26* for details.
BIR	Yes	This property holds the biometric data that is captured and returned to the application.
CapPrematchData	Partial	This property is initialized to false in the **open** method. It is not modified during execution of the application since the MOC (Match-on-Card) SmartCard technology needed to generate a processed **BIR** based on pre-match data stored on a SmartCard is not supported.
CapRawSensorData	Yes	This property is initialized to true in the **open** method and is not modified during execution of the application.
CapRealTimeData	Partial	This property is initialized to false in the **open** method and is not modified during execution of the application.
CapSensorColor	Yes	This property is initialized to BIO_CSC_GRAYSCALE in the **open** method and is not modified during execution of the application.

Name	Implemented[1]	Comment
CapSensorOrientation	Yes	This property is initialized to BIO_CSO_INVERTED in the **open** method and is not modified during execution of the application.
CapSensorType	Yes	This property is initialized to BIO_CST_FINGERPRINT in the **open** method and is not modified during execution of the application.
CapTemplateAdaptation	Partial	This property is initialized to false in the **open** method. It is not modified during the execution of the application since template adaptation is not supported.
RawSensorData	Yes	This property holds the raw biometric data that is captured and returned to the application.
RealTimeDataEnabled	Partial	This property is initialized to false in the **open** method and is not modified during execution of the application.
SensorBPP	Yes	This property is initialized by invoking the DigitalPersona function that defines the image resolution, in pixels.
SensorColor	Yes	This property is initialized by invoking the DigitalPersona function that defines the image type.
SensorHeight	Yes	This property is initialized by invoking the DigitalPersona function that defines the image height, in pixels.
SensorOrientation	Yes	This property is initialized to BIO_SO_INVERTED in the **open** method and is not modified during execution of the application.
SensorType	Yes	This property is initialized to BIO_ST_FINGERPRINT in the **open** method and is not modified during execution of the application.
SensorWidth	Yes	This property is initialized by invoking the DigitalPersona function that defines the image width, in pixels.
Common Methods		
Open	Yes	
Close	Yes	
ClaimDevice	Yes	
ReleaseDevice	Yes	
CheckHealth	No	
ClearInput	No	

Name	Implemented[1]	Comment
ClearInputProperties	No	
ClearOutput	No	This method is not supported by the Biometrics device category.
DirectIO	No	
CompareFirmwareVersion	No	
ResetStatistics	No	
RetrieveStatistics	No	
UpdateFirmware	No	
UpdateStatistics	No	
Specific Methods		
beginEnrollCapture	Yes	
beginVerifyCapture	Yes	
endCapture	Yes	
identify	Yes	
identifyMatch	Yes	
processPrematchData	No	
verify	Yes	
verifyMatch	Yes	

Events		
DataEvent	Yes	Valid values for *Status* parameter are 1- BIO_DATA_ENROLL if enroll capture is completed. 2- BIO_DATA_VERIFY if verify capture is completed.

Name	Implemented[1]	Comment
DirectIOEvent	Specific	Return values are 1 - DP_DIOE_DISCONNECT 2 - DP_DIOE_RECONNECT 3 - DP_DIOE_FINGER_TOUCHED 4 - DP_DIOE_FINGER_GONE 5 - DP_DIOE_IMAGE_READY 6 - DP_DIOE_SAMPLE_QUALITY 7 - DP_DIOE_ENROLL_FEATURES_ADDED 8 - DP_DIOE_OPERATION_STOPPED
ErrorEvent	Yes	**ResultCode** - Standard OPOS Result Code
	Specific	**ResultCodeExtended** - DigitalPersona Result Code
	Yes	**ErrorLocus** - EL_INPUT
	Yes	**ErrorResponse** - ER_CLEAR
OutputCompleteEvent	No	This event is not supported by the Biometrics device category.
StatusUpdateEvent	Partial	Valid values for *Status* parameter are 1 - BIO_SUE_RAW_DATA

1. Yes = Fully implemented
   No = Not implemented
   Partial = Implemented, but not all features are available
   Specific = Implemented with device-specific features or data

Exceptions

The files `jpos.Const.*` contain the JavaPOS standard exception codes. The following exceptions are thrown by the U.are.U JavaPOS API:

- JPOS_E_FAILURE
- JPOS_E_ILLEGAL
- JPOS_E_NOHARDWARE
- JPOS_E_TIMEOUT

Device-Related Error Codes

For the device-related result codes returned by *EventNumber* after the **DirectIOEvent** event, see *EventNumber Return Values* on *page 28*.

The following error codes are returned in the *ResultCodeExtended* property of the **ErrorEvent** object.

IMPORTANT: You should always use the names of the return codes, such as FT_OK, rather than the numeric values because these values may change at any time, without notice. The numbers are provided as a reference for the error codes in the sample application.

Value	Result Code	Description
-1	FT_ERR_NO_INIT	The fingerprint feature extraction module or the fingerprint comparison module is not initialized.
-2	FT_ERR_INVALID_PARAM	One or more parameters are not valid.
-4	FT_ERR_IO	A generic I/O file error occurred.
-7	FT_ERR_NO_MEMORY	There is not enough memory to perform the action.
-8	FT_ERR_INTERNAL	An unknown internal error occurred.
-10	FT_ERR_UNKNOWN_DEVICE	The requested device is not known.
-11	FT_ERR_INVALID_BUFFER	A buffer is not valid.
-33	FT_ERR_UNKNOWN_EXCEPTION	An unknown exception occurred.

HERE IS THE TABLE FROM PREV OPOS DOC:

The table below defines DigitalPersona device-related result codes returned to the *ResultCodeExtended* parameter of the **ErrorEvent** event.

Value	Result Code	Description
0	FT_OK	The function succeeded.
1	FT_WRN_NO_INIT	The fingerprint feature extraction module or the fingerprint comparison module are not initialized.
8	FT_WRN_INTERNAL	An internal error occurred.
9	FT_WRN_KEY_NOT_FOUND	The fingerprint feature extraction module or the fingerprint comparison module could not find an initialization setting.
11	FT_WRN_UNKNOWN_DEVICE	The fingerprint reader is not known.
12	FT_WRN_TIMEOUT	The function has timed out.
-1	FT_ERR_NO_INIT	The fingerprint feature extraction module or the fingerprint comparison module is not initialized.
-2	FT_ERR_INVALID_PARAM	One or more parameters are not valid.
-3	FT_ERR_NOT_IMPLEMENTED	The called function was not implemented.
-4	FT_ERR_IO	A generic I/O file error occurred.
-7	FT_ERR_NO_MEMORY	There is not enough memory to perform the action.
-8	FT_ERR_INTERNAL	An unknown internal error occurred.
-9	FT_ERR_BAD_INI_SETTING	Initialization settings are corrupted.
-10	FT_ERR_UNKNOWN_DEVICE	The requested device is not known.
-11	FT_ERR_INVALID_BUFFER	A buffer is not valid.
-16	FT_ERR_FEAT_LEN_TOO_SHORT	The specified fingerprint feature set or fingerprint template buffer size is too small.
-17	FT_ERR_INVALID_CONTEXT	The given context is not valid.
-29	FT_ERR_INVALID_FTRS_TYPE	The feature set purpose is not valid.
-32	FT_ERR_FTRS_INVALID	Decrypted fingerprint features are invalid. Decryption may have failed.
-33	FT_ERR_UNKNOWN_EXCEPTION	An unknown exception occurred.

Application Notes 13

This chapter describes some suggestions for troubleshooting your application.

General Fingerprint Issues

It is very important that you test your application with multiple and diverse people. The readability of fingerprints is affected by many factors including wear and tear, skin dryness, and age.

The middle finger often gives better scans than the index finger because the middle finger is typically less worn. We recommend that you enroll more than one finger for each user, preferably at least the index and middle fingers.

C/C++ Issues

The `dpfj_identify` function returns `DPFJ_SUCCESS` if it is able to do identification (i.e., the FMDs are valid and correctly formed). However that does not mean that a candidate was found. You must check the number of returned candidates to see if any actual matches were found.

The `dpfj_compare` function returns `DPFJ_SUCCESS` if it is able to do the requested comparison (i.e., the FMDs are valid and correctly formed). However that does not mean that the fingerprints matched. To check whether they matched, you must look at the dissimilarity score (0=no match, `maxint`=perfect match, `maxint-threshold` = acceptable match).

Glossary 14

General Terms

Fingerprint

The impression left from the friction ridges of a human finger.

Fingerprint characteristics

The biological finger surface details that can be detected and from which distinguishing and repeatable *fingerprint features* can be extracted.

Fingerprint minutiae

The *fingerprint characteristics* commonly used in fingerprint recognition systems. *Fingerprint minutiae* include:

- Ridge ending – the abrupt end of a ridge
- Ridge bifurcation – a single ridge that divides into two ridges
- Short ridge, or independent ridge – a ridge that commences, travels a short distance and then ends
- Island – a single small ridge inside a short ridge or ridge ending that is not connected to all other ridges
- Ridge enclosure – a single ridge that bifurcates and reunites shortly afterward to continue as a single ridge
- Spur – a bifurcation with a short ridge branching off a longer ridge
- Crossover or bridge – a short ridge that runs between two parallel ridges
- Delta – a Y-shaped ridge meeting
- Core – a U-turn in the ridge pattern.

Fingerprint Data

Fingerprint sample

An analog or digital representation of a *fingerprint* obtained from a *fingerprint capture device*. See also *fingerprint image*.

Fingerprint image

A digital representation of a *fingerprint sample* encoded as a spatially mapped array of pixels. A *fingerprint image* is the only representation of a *fingerprint sample* supported by DigitalPersona products, thus the terms *fingerprint image* and *fingerprint sample* are used interchangeably.

Fingerprint features

The digital representation of *fingerprint characteristics*.

Fingerprint Image Data (FID)

The binary data containing one or more *fingerprint images* from one or multiple fingers of the same person.

Fingerprint Image View (FIV)

The part of an *FID* that contains a *fingerprint image* from a single impression of a single finger.

In the majority of cases, an *FID* contains only one *FIV*.

Fingerprint Minutiae Data (FMD)

The digital representation of *fingerprint minutiae*, and optionally other *fingerprint characteristics*, from one or multiple fingers of the same person.

Fingerprint Minutiae View (FMV)

The part of an *FMD* that contains *fingerprint features* from a single impression of a single finger.

Typically *FMDs* contain only one *FMV*.

Fingerprint template

The *fingerprint minutiae data* that is stored as a result of the *enrollment* process.

Fingerprint Devices

Fingerprint capture device

A device that collects a signal of *fingerprint characteristic* and outputs a *fingerprint sample.* This device can consist of one or more components, including hardware and supporting software. For example, a *fingerprint capture device* may include a camera, photographic paper, a printer, and/or a digital scanner.

Fingerprint reader

A *fingerprint capture device* that obtains a *fingerprint image* by direct interaction with a finger.

Fingerprint Recognition Terms

Capture

The process of acquiring a *fingerprint image* from a *fingerprint reader* or *fingerprint capture device*.

Enrollment

The process of *capturing* a *fingerprint image*(s) for an individual, extracting *fingerprint features,* optionally checking for duplicates, and storing the *fingerprint features* (*fingerprint template*). See also *FMD, Fingerprint Template*.

Comparison

The function which, given two *fingerprints* computes a *comparison score*.

Comparison score

A *comparison score* is a numerical value resulting from the comparison of the *fingerprint features* of two *fingerprints.*

Comparison scores are of two types: *similarity scores* and *dissimilarity scores*.

Comparison scores are calculated using algorithms that are specific to the fingerprint recognition system and, optionally, can be normalized in order to maintain a constant score distribution of *impostor verification attempts* or *open set identification attempts*.

Similarity Score

See *comparison score.*

Dissimilarity Score

See *comparison score.*

Fingerprint Recognition

Verification or *identification*.

Verification

The process of

1. *Capturing* a *fingerprint image* from an individual,
2. Extracting *fingerprint features*,
3. *Comparing* the captured image's *fingerprint features* with the *fingerprint template*(s) of the enrollee this individual claims to be and
4. Making the *match/non-match decision.*

Verification Threshold

The maximum degree of dissimilarity that is allowed between a *fingerprint image* and a *fingerprint template* in order to make the *match decision* during the *verification* process.

Match decision or *match*

A decision that two *fingerprints* are from the same finger (meet the *verification threshold*).

Non-match decision or *non-match*

A decision that two *fingerprints* are not from the same finger (do not meet the *verification threshold).*

Identification

The process of

1. *Capturing* a *fingerprint image* from an individual,
2. Extracting *fingerprint features* and

3. *Comparing* the captured image's *fingerprint features* with the *fingerprint templates* from multiple individuals in order to create a candidate list of fingerprints that meet the *identification threshold*.

Identification Threshold

The maximum degree of dissimilarity that is allowed between a *fingerprint image* and a *fingerprint template* in order to call the *fingerprint template* a *candidate* in the *identification* process.

Candidate

A *fingerprint template* that is determined to be similar to a given *FMD* during identification.

Recognition Accuracy

The terminology below is used when discussing testing processes and reporting accuracy for fingerprint-enabled applications.

Genuine verification attempt

An attempt to compare *fingerprint features* with a *fingerprint template* where the *fingerprint features* and *fingerprint template* are from the same individual, and the *fingerprint features* and *fingerprint template* were captured at different times (recommended at least 2-3 weeks apart).

Impostor verification attempt

An attempt to compare *fingerprint features* with a *fingerprint template,* where the *fingerprint features* and *fingerprint template* are from two different people.

False match rate (FMR)

The ratio between the number of *impostor verification attempts* that produced a *match decision* and the total number of *impostor verification attempts.* See also *FAR.*

False non-match rate (FNMR)

The ratio between the number of *genuine verification attempts* that produced a *non-match decision* and the total number of *genuine verification attempts*. See also *FRR.*

False Accept Rate (FAR)

The application-specific measure of how often the application falsely grants a request.

In authentication applications, *FAR* is used in place of *FMR*.

False Reject Rate (FRR)

The application-specific measure of how often the application falsely rejects a request.

In authentication applications, *FRR* is used in place of *FNMR*.

Verification Detection Error Trade-off (DET) curve

A parametric curve that plots *FMR* on the x-axis and *FNMR* on the y-axis as a function of the *verification threshold*.

Probe

In simulation tests, a *fingerprint* used in searches (analogous to the user *fingerprint* used by a real application for searching).

Gallery

In simulation tests, a set of enrolled *fingerprints* (*fingerprint templates*) used for comparison purposes (analogous to the database of enrolled *fingerprint templates* used by a real application).

Open set identification attempt

For the purpose of testing, an attempt to compare a *probe* against a *gallery* that does not contain any impressions from any finger of the individual whose finger was used as the *probe*.

Closed set identification attempt

For the purpose of testing, an attempt to compare a *probe* against a *gallery*, where the *gallery* includes a *fingerprint template* from the same finger (from the same person) as used for the probe. The *probe* and the corresponding *fingerprint template* should be captured at different times (recommended at least 2-3 weeks apart).

False positive identification rate (FPIR)

Proportion of *identification* requests that return a *candidate* even though the person is not enrolled.

More precisely, for the purpose of recognition accuracy testing, the ratio between the number of *open set identification attempts* that returned one or more *candidates* and the total number of *open set identification attempts*.

False negative identification rate (FNIR)

Proportion of *identification* requests by enrollees that do not return the correct *candidate*.

More precisely, for the purpose of recognition accuracy testing, the ratio between the number of *closed set identification attempts* that did not return the correct *candidate* and the total number of *closed set identification attempts*.

Identification detection error trade-off (DET) curve

A parametric curve that plots *FPIR* on the x-axis and *FNIR* on the y-axis as a function of the *identification threshold*. The *Identification DET curve* depends on the *gallery* size. Typically, multiple *identification DET curves* are shown on the same plot, one for each *gallery* size.